Ph.D. DISSERTATION

Design of Energy Efficient Neural Networks

에너지 효율적 인공신경망 설계

BY

Jaehyun Kim

February 2019

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Design of Energy Efficient Neural Networks

에너지 효율적 인공신경망 설계

BY

Jaehyun Kim

February 2019

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Design of Energy Efficient Neural Networks

에너지 효율적 인공신경망 설계

지도교수 최 기 영
이 논문을 공학박사 학위논문으로 제출함

2019년 2월

서울대학교 대학원

전기정보공학부

김 재 현

김재현의 공학박사 학위 논문을 인준함

2019년 2월

위 원 장:	김 태 환	(인)
부위원장:	최 기 영	(인)
위 원:	유 승 주	(인)
위 원:	이 재 욱	(인)
위 원:	김 경 훈	(인)

Abstract

Recently, deep learning has shown astounding performances on specific tasks such as image classification, speech recognition, and reinforcement learning. Some of the state-of-the-art deep neural networks have already gone over human's ability. However, neural networks involve tremendous number of high precision computations and frequent off-chip memory accesses with millions of parameters. It incurs problems of large area and exploding energy consumption, which hinder neural networks from being exploited in embedded systems. To cope with the problem, techniques for designing energy efficient neural networks are proposed.

The first part of this dissertation addresses the design of spiking neural networks with weighted spikes which has advantages of shorter inference latency and smaller energy consumption compared to the conventional spiking neural networks. Spiking neural networks are being regarded as one of the promising alternative techniques to overcome the high energy costs of artificial neural networks. It is supported by many researches showing that a deep convolutional neural network can be converted into a spiking neural network with near zero accuracy loss. However, the advantage on energy consumption of spiking neural networks comes at a cost of long classification latency due to the use of Poisson-distributed spike trains (rate coding), especially in deep networks.

We propose to use weighted spikes, which can greatly reduce the latency by assigning a different weight to a spike depending on which time phase it belongs. Experimental results on MNIST, SVHN, CIFAR-10, and CIFAR-100 show that the proposed spiking neural networks with weighted spikes achieve significant reduction in classification latency and number of spikes, which leads to faster and more energy-efficient spiking neural networks than the conventional spiking neural networks with rate coding. We also show that one of the state-of-the-art networks the deep residual network

can be converted into spiking neural network without accuracy loss.

The second part of this dissertation focuses on the design of highly energy-efficient analog neural networks in the presence of variations. Analog hardware accelerators for deep neural networks have taken center stage in the aspect of high parallelism and energy efficiency. However, a critical weakness of the analog hardware systems is vulnerability to noise. One of the biggest noise sources is a process variation. It is a big obstacle to using analog circuits since the variation shifts various parameters of analog circuits from the correct operating points, which causes severe performance degradation or even malfunction.

To achieve high energy efficiency with analog neural networks, we propose resistive random access memory (ReRAM) based analog implementation of binarized neural networks (BNNs) with a novel variation compensation technique through activation matching (VCAM). The proposed architecture consists of 1-transistor-1-resistor (1T1R) structured ReRAM synaptic arrays and differential amplifier based neurons, which leads to high-density integration and energy efficiency. To cope with the vulnerability of analog neurons due to process variation, the biases of all neurons are adjusted in the direction that matches average output activation of ideal neurons without variation. The technique effectively restores the classification accuracy degraded by the variation. Experimental results on 32nm technology show that the proposed architecture achieves the classification accuracy of 98.55% on MNIST and 89.63% on CIFAR-10 in the presence of 50% threshold voltage variation and 15% resistance variation at 3-sigma point. It also achieves 970 TOPS/W energy efficiency with MLP on MNIST.

keywords: Deep neural network, spiking neural network, weighted spike, supervised learning, analog neuron, ReRAM, activation matching, variation compensation

student number: 2015-30195

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Deep Neural Networks with Weighted Spikes

Nowadays deep neural networks (*DNN*s) are continuously expanding their influences on application areas such as image classification [9], speech recognition [10], natural language processing [11] and many others. However, its heavy computational load and high energy consumption still block the broader use of DNNs in practical applications that require large-scale data processing in real-time [12]. As an alternative, spiking neural networks (*SNN*s) have been studied by many researchers for the purpose of building neuromorphic hardware [13–17] with low energy consumption. A spiking neural network consists of spiking neurons, each of which fires an output spike only when its membrane potential is charged above a certain threshold [18]. The generated spike is propagated into the neurons in the next layer and increases/decreases their membrane potentials. In this manner, the communication between neurons is performed by spikes. In SNNs, the processing of each input spike in a neuron accompanies only a single simple addition operation onto the membrane potential, while conventional artificial neural networks (*ANNs*) require multi-bit input signals and a multiplication operation in addition to accumulation, which consumes much larger energy.

There are various approaches to train a spiking neural network. Among those, one popular way is to train an ANN with the same topology and then convert the synaptic weights to those of the SNN. The resulting SNN achieves high classification accuracy comparable to the ANN even for a deep topology. Most of the ANN-to-SNN conversion approaches use Poisson-distributed rate coding [19], where spike firing rate or the number of spikes generated within a certain time interval approximates the signal intensity. Rate coding inevitably requires a long time to represent high precision information, which means it has a low information capacity. Because of this, the classification latency increases much longer if the depth of SNN increases since spikes can be propagated into next neurons only after the membrane potentials of the current neurons are charged over the given threshold. Moreover, the information represented

by the spike firing rate becomes more error-prone in deeper layers of the network [20], and thus larger number of spikes is required to reduce the approximation errors in deep neural networks. Since the number of additions is proportional to the number of spikes arriving at neurons in SNNs, larger number of spikes incurs more dynamic energy consumption and significantly diminishes the merit of low energy consumption in deep SNNs.

To overcome the problem, we propose a deep spiking neural network with weighted spikes (*SNN-WS*)[1] to perform the image classification with shorter classification latency and less number of spikes compared to the SNN with conventional Poisson-distributed rate coding (*SNN-RC*).[2] It assumes that a neuron can fire at most one spike within a time step. It assigns different weights to spikes depending on their phases (relative position of their time steps within a period) in order to transfer more information to the deeper layers in a short time. This scheme is partly inspired by the phase coding [22, 23] in neural coding field of neuroscience. Exploiting the characteristics of the weighted spikes, we also propose an early decision algorithm to take the trade-off between classification accuracy and energy. We validate the proposed idea through experiments performed on various datasets such as MNIST, SVHN [24], CIFAR-10, and CIFAR-100 [25] with various network topologies.

The contributions of this paper are enumerated below.

- Proposing a novel spiking neural network model with weighted spikes inspired by phase coding, which has shorter classification latency and lower energy consumption than conventional SNN-RC.
- Providing a mathematical formulation showing that SNN-WS approximates the ReLU activations of ANN.
- Devising an update skipping scheme that reduces the effect of noise spikes and increases the convergence speed.

[1] Note that the weight in this context does not mean synaptic weight.

[2] This work is published in Neurocomputing 2018 [21].

- Combining an early decision scheme for further energy saving by allowing a small accuracy loss.
- Extensive experimental results on MNIST, SVHN, CIFAR-10, and CIFAR-100 datasets.

1.2 VCAM: Variation Compensation through Activation Matching for Analog Binarized Neural Networks

Recently, deep learning has shown astounding performances on specific tasks such as image classification [9], speech recognition [10], and reinforcement learning [26]. In image classification tasks, some of the state-of-the-art deep neural networks have already gone over human's ability [27]. However, neural networks involve tremendous number of high precision computations and frequent off-chip memory accesses with millions of parameters. It incurs problems of large area and exploding energy consumption, which hinder neural networks from being exploited in embedded systems for real-time purpose.

To cope with these problems, the approaches to reduce the size of weights and activations are introduced for smaller computation complexity and data storage [1, 2]. The authors in [1] firstly have shown that the feed forward of deep neural networks can be operated with only marginal accuracy loss by using ternary weights (-1, 0, 1) and 1-bit activations. They also proposed a retraining method during backpropagation which uses both high-precision weights for proper accumulations of small gradients and low-precision weights for computationally efficient feed forward. The weight and activation sizes are further reduced in [2]. Binarized neural network (BNN) [2] uses binary weights (-1 and +1) and binary activations (-1 and +1) instead of the conventional 32-bit floating point weights and activations for inference, and it incurs only marginal accuracy loss. The use of binary weights and activations reduces the size of parameters directly and replaces a lot of multiply-accumulate (MAC) operations with simple bit-wise XOR and bit counting operations, which drastically reduces the area and power requirements for parameter storage (memory and buffers) and computational units. Due to the advantages of area and power reductions, BNN has been implemented on many different hardware platforms such as GPU [2], FPGA [28], and ASIC with digital and analog processing elements [6–8, 29–32].

In many cases, data movements between the core and off-chip memory consume higher energy than computations in the core [33], and thus on-chip SRAM or embedded DRAM memory is exploited in neural network hardware accelerators [6, 31, 34]. However, such on-chip memory still has low cell density and high power consumption [35] prohibiting the design of large scale networks. In contrast, resistive random access memory (ReRAM) aiming at on-chip memory is non-volatile memory (NVM) with near-zero leakage energy consumption and high cell density. Even more, it supports operations such as matrix-vector-multiplication (MVM) and bit-wise operations within memory [33]. Due to the advantages, the BNN implementations with ReRAM show outstanding energy and area efficiency [7, 29, 30, 32]. However, none of the previous works considered the severe non-linearity of transistors caused by process variation.

In this work, we propose a ReRAM-based analog implementation of BNN and a novel variation compensation technique (the technique is applicable to any analog implementations other than the ReRAM-based one). The contributions of this paper are as follows:

- A BNN architecture with 1-transistor-1-resistor (1T1R) structured ReRAM synapses and analog neurons based on differential amplifiers (DAs) for high density integration, fast classification speed, and low energy consumption.
- A modified BNN algorithm that merges the parameters of batch normalization layers into biases, which reduces hardware implementation cost without accuracy loss.
- A novel variation compensation through activation matching (VCAM) technique that adjusts the biases of all neurons to match the average activation of each neuron in the analog implementation to that in the trained model, which effectively restores the classification accuracy degraded by process variation.
- Analysis of the variation compensation technique under different process varia-

tions by using Monte Carlo simulation with 32nm technology library.

- Achieved accuracy of 98.55% for MLP on MNIST and 89.63% for CNN on CIFAR-10 under process variation, and 970 TOPS/W energy efficiency for MLP on MNIST.

Chapter 2

Background

2.1 Spiking neural network

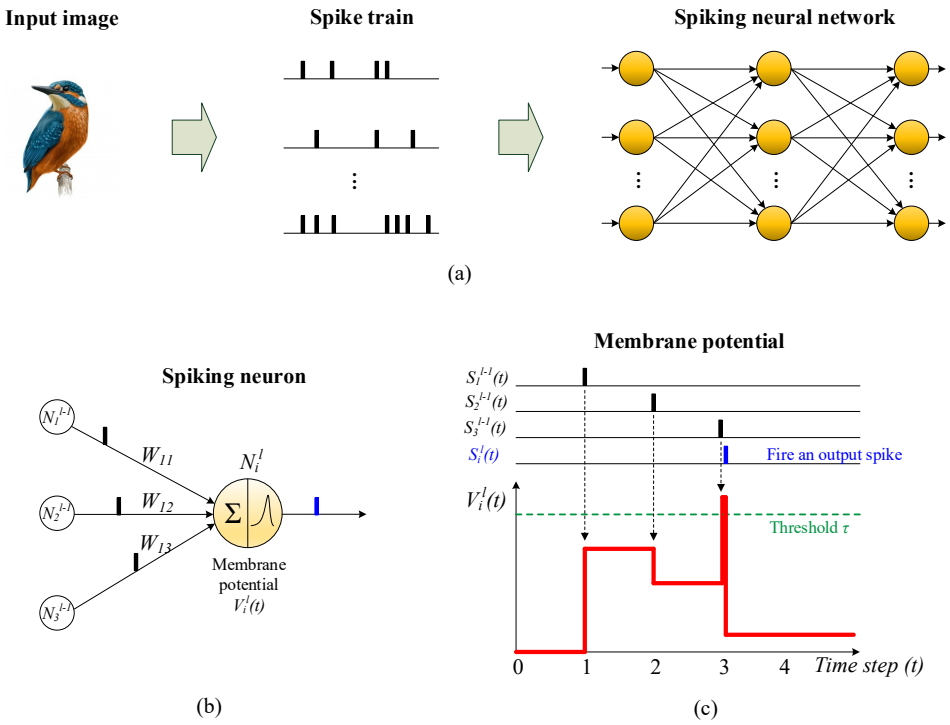


Figure 2.1: Example of a spiking neural network. Each pixel of an image is encoded into a spike train, and the resulting spike trains are fed into the spiking neural network. The network consists of spiking neurons, each of which integrates postsynaptic potentials and fires a new spike if its membrane potential exceeds a certain threshold.

Spiking neural network is a class of artificial neural networks that emulates the behavior of a brain with spiking neurons. In a brain, neurons communicate with spike signals. An incoming spike is mediated by a synapse, and the synapse produces a postsynaptic potential (*PSP*) which is affected by the weight of the synapse. The postsynaptic potential then changes the membrane potential, the internal state of a neuron. The membrane potential can be represented as the sum of the PSPs generated from all synapses that receive input spikes. If the membrane potential goes above a certain

threshold, the neuron fires a new spike and it is propagated into other neurons.

Figure 2.1 demonstrates a process of image classification by a spiking neural network. First, spike trains are generated according to an input image. Each pixel in an image is encoded as a spike train based on its intensity, and the resulting spike trains are applied as an input of the SNN. When spikes are propagated through synapses, PSPs are induced with different intensity based on synaptic weights. A spiking neuron integrates all incoming PSPs into its membrane potential, and fires a new spike if the membrane potential exceeds a certain threshold. With this mechanism, input spike trains lead to new spike trains at intermediate layers of the SNN, and a classification decision can be made by comparing the firing rate or timing of the output spikes.

The main advantage of SNNs compared to conventional ANNs is high energy efficiency. While a neuron of an ANN requires high precision multiplications of input values and synaptic weights, that of an SNN requires only additions or integrations of synaptic weights into its membrane potential only at the presence of spikes. Due to the operational simplicity, it can be implemented with low energy consumption. The energy consumption required to handle a spike is just a few pJ in current SNN implementations [13–17], and TrueNorth chip [15], a popular SNN implementation, consumes only 72mW for 1 million neurons. Moreover, the in-situ placement of synapses near neurons in SNNs avoids huge energy consumption required for external memory access in ANN implementations.

2.2 Spiking neuron model

To model the behavior of a spiking neuron, we use a simple *integrate-and-fire model* that is similar to the previous works [20, 36]. In the model, we define An occurrence of an output spike by the i-th neuron in layer l at time t as

$$s_i^l(t) = \begin{cases} 1, & \text{if there is a spike at time } t \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

An input current, the sum of postsynaptic potentials, which flows into the i-th neuron in layer l is represented as

$$z_i^l(t) = \sum_{j=1}^{M^{l-1}} W_{ij}^l s_j^{l-1}(t) + b_i^l, \tag{2.2}$$

where $s_j^{l-1}(t)$ is the output spike of j-th neuron in layer $l-1$ and b_i^l is the bias for the i-th neuron in layer l. Although it is not clear that there exists a bias in a biological neuron, the bias term is added to accurately convert a trained ANN to an SNN.

The membrane potential of a neuron, which gradually increases as it takes input currents induced by PSPs and decreases after generating outgoing spikes when it goes over a certain threshold τ, can be represented as

$$V_i^l(t) = V_i^l(t-1) + z_i^l(t) - \tau s_i^l(t). \tag{2.3}$$

This is called a *reset by subtraction model* in which a membrane potential decreases by the amount of the threshold level τ when it fires an output spike. We do not model membrane potential leakage that makes the membrane potential decrease over time, and the refractory period that makes the membrane potential settle into a base level after firing a spike. It is because an adoption of these factors could rather deteriorate classification accuracy and incur computational overhead.

2.3 Rate coding in SNNs

A rate coding is one of the most popular encoding schemes used in conventional SNNs to represent a signal with a spike train. In the rate coding scheme, a spike firing frequency of a neuron represents a signal intensity. The spike firing rate r_i of i-th neuron can be formulated as

$$r_i = \frac{N_s}{T} = \frac{\sum_{t=1}^{T} s_i(t)}{T}, \tag{2.4}$$

when N_s spikes are fired during T time steps. There can be a spike or no spike in each time step. The firing rate is 1.0 if a neuron fires every time step, and the firing rate is 0.0 if it fires nothing for all time steps. In an image classification task, for example, if a pixel intensity of an image is in the range of [0.0, 1.0], it can be represented by a spike train with the firing rate of [0.0, 1.0] as an input to the SNN.

The Poisson-distributed spike train [19] transmitted to i-th input neuron which corresponds to a pixel intensity P can be generated by comparing a random number and P independently for each time step as

$$s_i^0(t) = \begin{cases} 1, & \text{if Random(0.0, 1.0)} < P \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

The rate coding suffers from low information capacity. It requires many time steps to recognize a signal intensity correctly since the number of fired spikes has to be counted over the whole period. To recognize a signal of K-bit precision, it requires at least 2^K time steps. This intrinsic property of rate coding increases a classification latency and the number of spikes generated in a network. The increased number of spikes incurs higher dynamic energy consumption due to more frequent changes of membrane potentials, and the increased running time increases leakage energy consumption, which offsets the energy efficient nature of SNN.

2.4 Binarized neural networks

There has been many researches to reduce heavy computational overhead of deep neural networks. One of the approaches is weight and activation quantization method. Typically, 32-bit float point is used to represent weights and activations in neural networks. The quantization method reduces the sizes of weights and activations by using smaller data format such as 8-bit fixed point instead of the 32-bit floating point. Binarized neural network (BNN) is a result of extreme quantization, which uses 1-bit weights and 1-bit activations. The BNN and SNN are similar in that the 1-bit activation of the BNN looks like the spiking firing of the SNN.

The authors in [1] have shown that the feed-forward of deep neural networks can be performed by using ternary weights (-1, 0, and +1) with a fixed scale factor Δ and 1-bit activations as shown in Figure 2.2. The quantization step size Δ is determined with exhaustive search, and it is used in the retraining stage with error backpropagation. The feed-forward and backward gradient propagation are performed with the

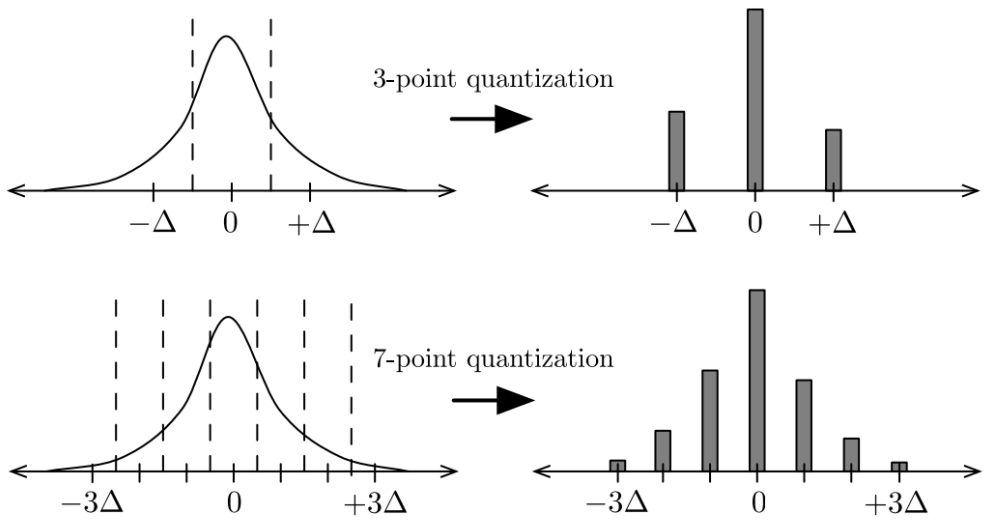


Figure 2.2: Histograms of weights before and after applying 3-point and 7-point symmetric uniform quantization where Δ denotes a quantization step size [1].

Definition of error signal:

$$\delta_i = -\frac{\partial E}{\partial net_i}$$

Forward step:

$$net_i = \sum_{j \in A_i} w_{ij}^{(q)} y_j^{(q)}$$

$$y_i^{(q)} = R_i\left(\phi_i\left(net_i\right)\right)$$

Backward step:

$$\delta_j = \phi_j'\left(net_j\right) \sum_{i \in P_j} \delta_i w_{ij}^{(q)}$$

Gradient calculation:

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j^{(q)}$$

Weight update:

$$w_{ij,new} = w_{ij} - \alpha \left\langle \frac{\partial E}{\partial w_{ij}} \right\rangle$$

$$w_{ij,new}^{(q)} = Q_{ij}\left(w_{ij,new}\right)$$

Figure 2.3: Summarization of the retraining algorithm in [1], where E is the output error, net_i is the summed input value of unit i, δ_i is the error signal of unit i, w_{ij} is the weight from unit j to unit i, y_j is the output signal of unit j, α is the learning rate, A_i is the set of units anterior to unit i, P_j is the set of units posterior to unit j, $R(\cdot)$ is the signal quantizer, $Q(\cdot)$ is the weight quantizer, and $\phi(\cdot)$ is the activation function. The superscript (q) indicates that the value is quantized and $< \cdot >$ averages the value over a mini-batch. Biases can be regarded as weights from the constant input value of 1.

quantized weights. However, the resulting gradients of backpropagation are too small to be directed applied to the quantized weights. Thus, high-precision weights are kept to accumulate the small gradients and they are quantized with Δ to produce the newly quantized weights. The experimental result on MNIST showed that the classification error of the feed-forward with ternary weights and 1-bit activations was 1.45%, which is only 0.48%p larger than the original classification error (0.97%).

The quantization approach is enhanced by the authors in [2]. Binarized neural network (BNN) [2] uses binary weights (-1 and +1) and binary activations (-1 and +1) instead of the conventional 32-bit floating point weights and activations for inference, and it incurs only marginal accuracy loss. The use of binary weights and activations re-

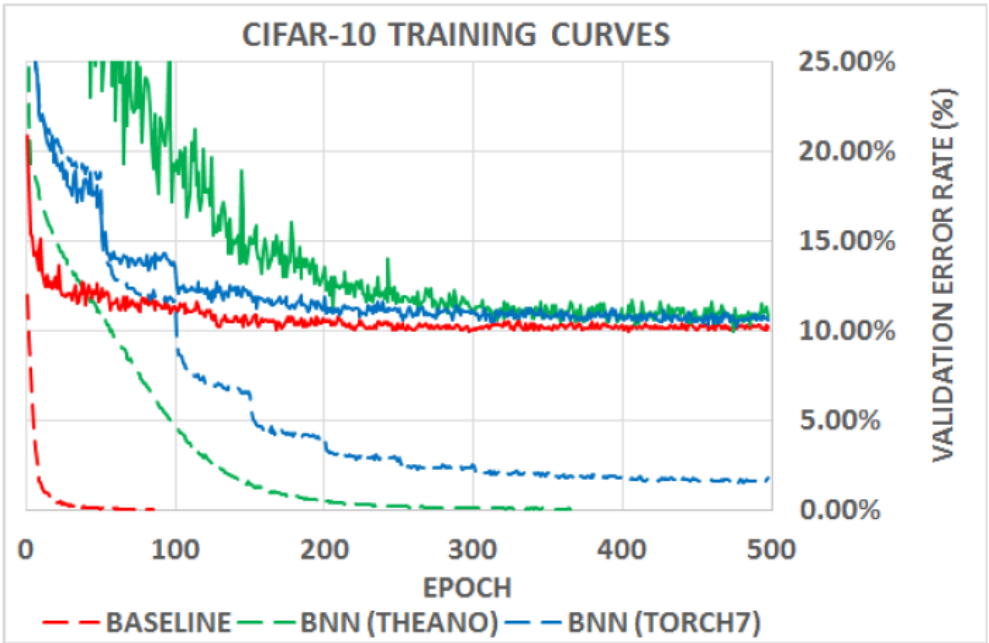


Figure 2.4: Training curves of a ConvNet on CIFAR-10 depending on the method in [2]. The dotted lines represent the training costs and the continuous lines the corresponding validation error rates. Although BNNs are slower to train, they are nearly as accurate as 32-bit float DNNs.

Table 2.1: Accuracy comparison between a DNN and BNNs

Dataset	Net	Configuration	Accuracy
MNIST	DNN	784-64-64-64-10	98.21%
	BNN (x1)	784-64-64-64-10	95.67%
	BNN (x2)	784-128-128-128-10	97.35%
	BNN (x4)	784-256-256-256-10	98.22%
	BNN (x8)	784-512-512-512-10	98.48%
CIFAR-10	DNN	32c-32c-s-64c-64c-s-128-10	86.88%
	BNN (x1)	32c-32c-s-64c-64c-s-128-10	74.15%
	BNN (x2)	64c-64c-s-128c-128c-s-256-10	82.13%
	BNN (x4)	128c-128c-s-256c-256c-s-512-10	86.51%
	BNN (x8)	256c-256c-s-512c-512c-s-1024-10	88.85%

duces the size of parameters directly and replaces a lot of multiply-accumulate (MAC) operations with simple bit-wise XOR and bit counting operations, which drastically reduces the area and power requirements for parameter storage (memory and buffers) and computational units. The BNN is trained in a similar manner proposed in [1]. It keeps the high-precision weights for the gradient accumulation and quantizes them to produce the binary weights (-1 or +1). The experimental results showed that the binarized MLP with 3 hidden layers (2048 neurons per layer) on MNIST achieved 1.40% classification error and the CNN on CIFAR-10 achieved 10.15% classification error as shown in Figure 2.4.

Although BNNs suffer from the degraded classification accuracy, it can be compensated by increasing the number of neurons in BNNs. Table 2.1 compares the classification accuracies of a baseline DNN and the same-topology BNNs with 1-8x increased number of neurons. For MNIST, MLP is used for the experiment. For CIFAR-10, CNN is used. The 'c' in the configuration column means a convolutional layer and

the 's' means a max sub-sampling layer. If the network topology is the same and the same number of neurons is used for both DNN and BNN, BNN experiences a loss of classification accuracy. However, the accuracy loss can be restored by increasing the number of neurons in every layer. According to our experiments, using 4x more neurons for a BNN leads to the similar classification accuracy of its corresponding DNN as shown in Table 2.1. If the number of neurons increases 4 times, the number of weights increases by a factor of 16. Nevertheless, the 1-bit weights and activations in BNNs require smaller data storage and cheaper computing units than 32-bit floating point weights and activations in DNNs. Thus, BNNs are still advantageous.

2.5 Resistive random access memory

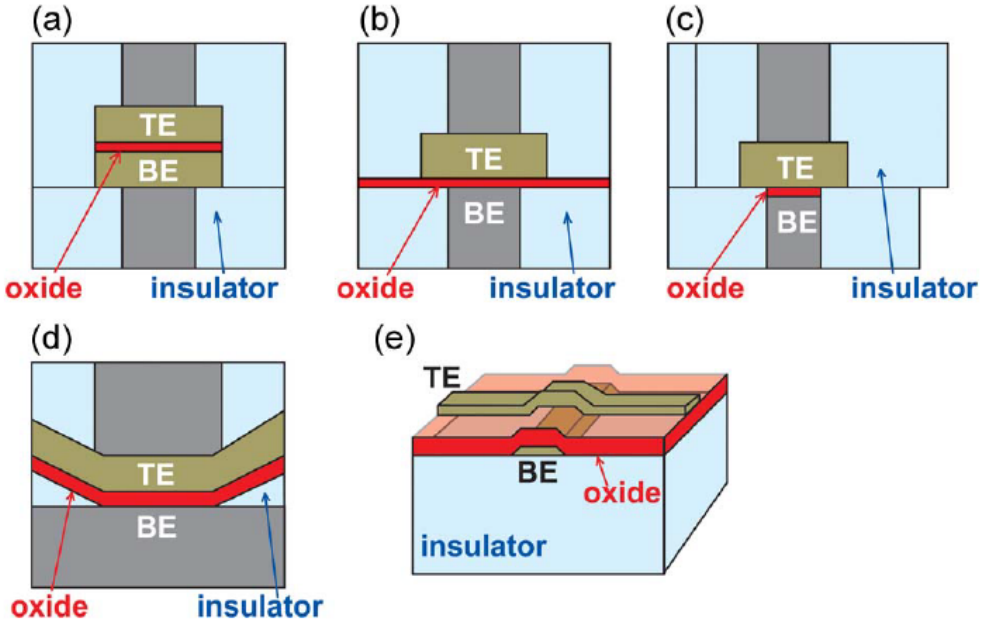


Figure 2.5: Different ReRAM structures [3]. (a) Typical Metal-Oxide-Metal (MOM) simple stacking structure of ReRAM. (b) The metallic via as a bottom electrode with oxide and top electrode layers. (c) The oxidized via material for the resistance switching oxide layer. (d) The concave structure. (e) The cross-bar structure consisting of the bottom and top electrode wires and blanket oxide film.

Resistive random access memory (ReRAM) is a non-volatile memory exploiting the characteristic of resistance change across metal oxide that can be adjusted by voltage control [3]. It is also called as a *memristor*. ReRAM is one of promising non-volatile memories due to its advantageous characteristics of the CMOS process compatibility and the small cell size ($4F^2$ at the minimum). The features are very useful for using ReRAM as a synaptic weight storage for neural network implementations since the high integration density and non-volatile memory features enable the on-chip integration of synaptic weights, which avoids heavy power consumption of off-chip

memory communications for the synaptic weight access.

The ReRAM consists of top electrode (TE), bottom electrode (BE), and the oxide layer between the two electrodes, which can be fabricated in metal layers. Figure 2.5 shows the illustrations of different ReRAM structures. Figure 2.5a shows the typical Metal-Oxide-Metal (MOM) structure made of two metallic electrodes and an oxide layer. The metallic via can be used as the bottom electrode (Figure 2.5b). If the diameter of the via is smaller than that of top electrode, the effective size of ReRAM is determined by the via diameter. Thus, by reducing the via diameter, ReRAM size can be shrunk. For the simpler fabrication process, the top side of via can be oxidized as Figure 2.5c. However, the choice of oxide material is limited in this structure. Figure 2.5d shows the concave structure of the insulating layer. The ReRAM size can be scaled down by decreasing the concave area. The cross-point structure with the blanket oxide film is illustrated in Figure 2.5e. In this case, the ReRAM size is defined by the width of TE and BE wires. The ReRAM cells can be stacked in any form of the structures, which leads to high integration density.

Figure 2.6a shows the unipolar operation of memristor. If the 2.3V is applied at first, the resistance goes down and the memristor state changes to a low-resistance state (LRS). This process is called *forming*. It is a result of a soft breakdown of the Metal-Oxide-Metal (MOM) structure. By sweeping the voltage with the same polarity as forming, the resistance can be increased to a high-resistance state (HRS), and this process is called *reset*. Once the memristor is formed, its state can go back to the LRS again by applying the voltage of 1.2V which is lower than the voltage required for the forming. This process is called *set*. In the bipolar operation, the voltage polarity is different between set and reset processes as shown in Figure 2.6b.

Figure 2.7 describes the forming, set, and reset operations. According to [37], during the forming process, soft dielectric breakdown occurs and oxygen ions drift to the anode interface by the high electric field, where they are discharged as neutral nonlattice oxygen if the anode materials are noble metals or react with the oxidizable anode

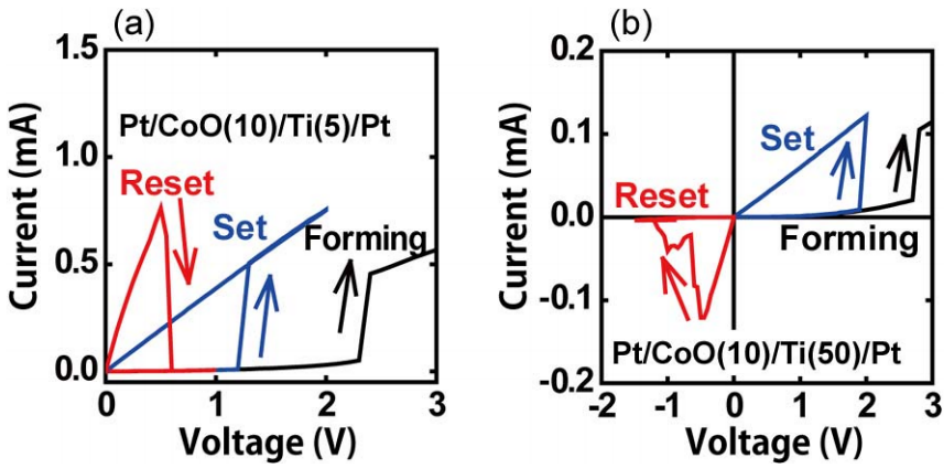


Figure 2.6: (a) Unipolar and (b) bipolar operations (forming, set, and reset) of CoO-based ReRAM with the bottom electrode of Pt and the top electrode consisting of Ti/Pt. The voltage sweep is done by the top electrode drive [3].

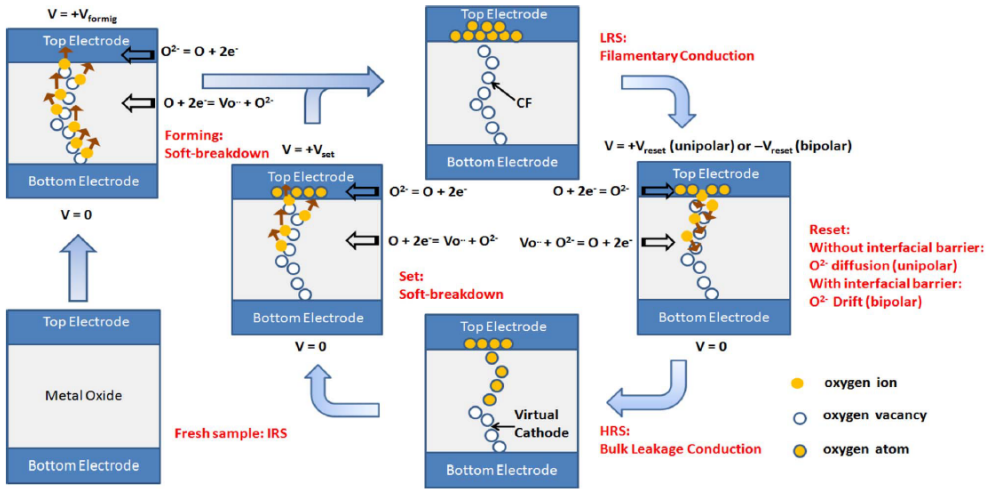


Figure 2.7: Schematic illustration of the switching process in the metal–oxide ReRAM. Adapted from [4].

materials to form an interfacial oxide layer. Thus, the electrode/oxide interface behaves like an oxygen reservoir [38]. For a memory cell in the LRS, the current flows through

the conductive filaments in the bulk oxide. During the reset process, oxygen ions migrate back to the bulk either to recombine with the oxygen vacancies or to oxidize the metal precipitates and return the memory cell to the HRS.

Chapter 3

Related Work

3.1 Training SNNs

Many approaches have been proposed for more efficient and accurate training of SNNs. However, it is not trivial and often suffers from a significantly low accuracy. One approach is an unsupervised learning by implementing spike timing-dependent plasticity (*STDP*) [39, 40], which is inspired by the learning behavior of biological neural networks in brains. However, the existing methods based on the STDP learning suffer from low classification accuracy and training difficulty on deep networks [41–45].

Another approach is a supervised learning by exploiting backpropagation algorithm [46], which is a very popular method used to train conventional ANNs. Many previous researches have applied the backpropagation algorithm in order to train SNNs, and most of them have achieved significantly better classification accuracy and training stability even in deep networks. The SNN researches using the backpropagation algorithm can be categorized into two groups.

One is to train synaptic weights of SNNs directly by using the spikes which are propagated in the networks. *SpikeProp* [47] is one of the first supervised learning algorithm for SNNs. It extends the traditional backpropagation algorithm to transfer the information in the timing of spikes. It shows that the learning algorithm can be used to solve real-world classification problems, and its learning efficiency has been improved by following researches [48–51]. However, they can control the timing of single-spike output only and limit the information capacity they can handle. To overcome the problem, a multi-spike learning method called remote supervised learning method (*ReSuMe*) is proposed [52]. It trains SNNs to produce desired output spike train by exploiting STDP and anti-STDP processes based on the Widrow-Hoff rule [53, 54]. *DL-ReSuMe* [55], which combines delay shifting on the ReSuMe, is proposed to enhance the learning performance further. The work in [56, 57] exploits perceptron neurons to improve training speed significantly, and the recent work in [58] proposes relative ordering learning (*ROL*) to remove the strict timing constraint of the previous work for more robust learning. These learning methods of direct spike handling

are shown to be successful for SNNs. However, only easy problems are solved by the methods and the networks used in the experiments are small and shallow compared to the networks used in conventional ANNs. There has been no attempt yet to show the feasibility of those methods for problems more complex than MNIST [59].

The other is to convert the synaptic weights of an ANN with the same topology to those of the SNN. The ANN-to-SNN conversion achieves accuracy nearly the same as that of the ANN even for deep networks and more complex datasets, because it uses synaptic weights well optimized by the backpropagation algorithm for ANNs. Early researches on the ANN-to-SNN conversion [60, 61] use more biologically realistic neurons with leakage and refractory periods, but suffer from significant accuracy loss. The work in [62] exploits ReLU activation function [63] to achieve more accurate results. The work in [36] achieves nearly lossless accuracy compared to ANNs on MNIST dataset. They propose an ANN-to-SNN conversion method by fixing the threshold of neurons to 1.0 and normalizing synaptic weights such that the maximum possible positive input to a neuron can only generate one spike to avoid the approximation error caused by activation values larger than 1.0. There is a similar weight normalization technique [20] tested on more complex CIFAR-10 [25] dataset. It analyzes the distribution of ReLU activation values and selects the 99.9th percentile value instead of the maximum activation as a normalization factor, which leads to faster classification.

3.2 SNNs with various spike coding schemes

Rate coding [19, 64] is the most frequently used coding technique for spikes in SNNs. Thus we use it as the control group in our experiments. In rate coding, the number of spikes occurred within a period of time is counted, and the spike firing rate is used as the signal intensity. As a result, rate coding has much redundancy and also tolerance to a certain amount of noise. For this reason, along with the fact that it fits well with the nature of SNNs, it is often used in many of SNN researches [20, 36, 60, 61]. However, it often suffers from a long classification latency.

While the rate coding drops out most of the timing information of the spikes, various approaches of temporal coding have been proposed to use timing information to mitigate the slowdown. For instance, time-to-first-spike [65] is a well-known temporal coding. In this technique, only the first spike within a certain time range has a meaning, and the arrival time of the first spike is used as the information. Time-to-first-spike coding is often used in many approaches [47, 66], because it is simple to understand, yet it can save useful information that is lost in rate coding. There are many other coding schemes, such as rank-order coding [67] and resonant burst model [68]. Among them, it would be worth mentioning phase coding [22, 23]. In phase coding, a background periodic oscillation function is defined as a reference. The value of a spike is determined by the phase of the reference function at the arrival time of the spike. Our work is conceptually close to phase coding because the spike weights are determined by the current phase.

There is a recent work that exploits burst spikes in order to build fast and efficient SNNs [5]. The authors compare three different spike encoding schemes such as rate coding, phase coding, and burst spiking as shown in Figure 3.1. The authors designed the burst spiking such that spikes have exponentially increasing weights in case of successive firing. For example, if a neuron fires three consecutive spikes, the first spike has a weight of 1, the second has a weight of 2, and the third has a weight of 4. Therefore, when a membrane potential of a neuron is large enough, the burst spiking

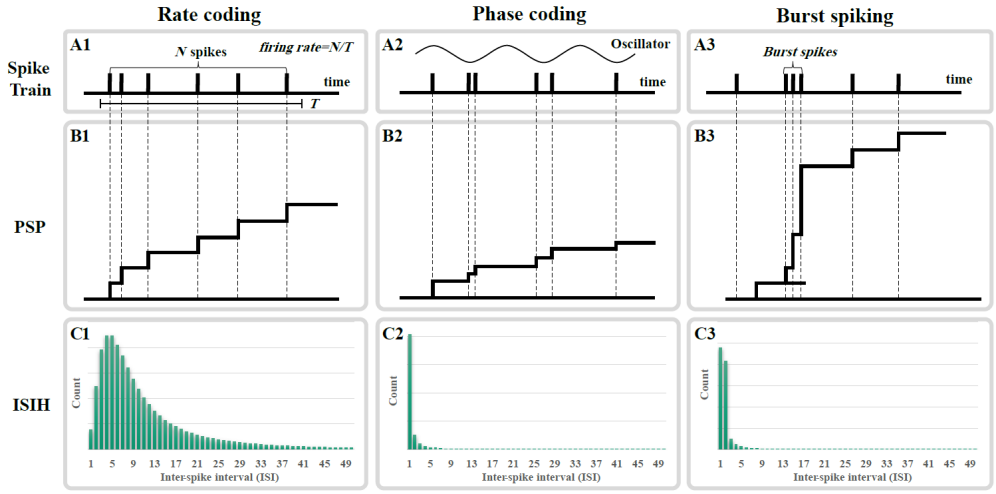


Figure 3.1: (A) Spike train, (B) post-synaptic potentiation (PSP), and (C) inter-spike interval histogram (ISIH) of the integrate-and-fire (IF) neurons depending on the various types of neural coding. Adapted from [5].

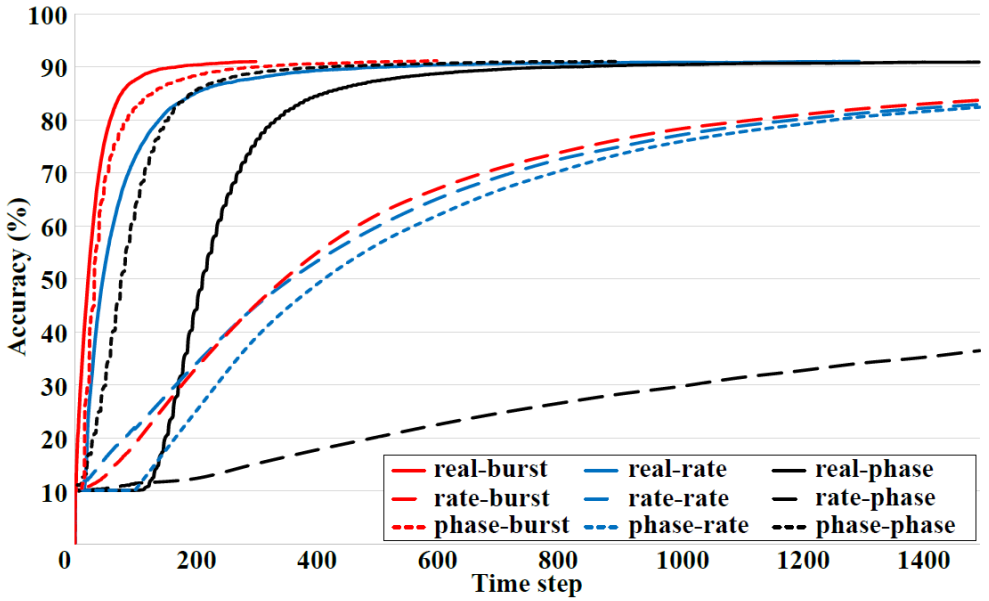


Figure 3.2: Inference curve of various neural coding schemes adapted from [5].

encoding scheme makes the network transfer much information in a short time, and it increases the computation speed of SNNs. Figure 3.2 shows the inference curves of various combination of spike encoding schemes separately applied for an input layer and hidden layers. The *real* means that analog signal is fed into the neurons in the first layer, which means that the neurons in the first layer are ideal neurons instead of spiking neurons. Except for the *real-burst* case which uses ideal neurons in the first layer, the *phase-burst* case shows the shortest inference latency.

3.3 BNN implementations

Due to the advantages of area and power reductions caused by 1-bit weights and 1-bit activations, BNN has been implemented on many different hardware platforms such as GPU [2], FPGA [28], and ASIC with digital and analog processing elements [6–8, 29–32].

The authors in [6] proposed a BNN architecture consisting of SRAM synaptic weights and switched capacitor based analog neurons. Figure 3.3 represents the overall architecture. The weights stored in SRAMs input data are loaded onto flip-flops and the following XOR gates perform the XOR operations. The analog adders produce the

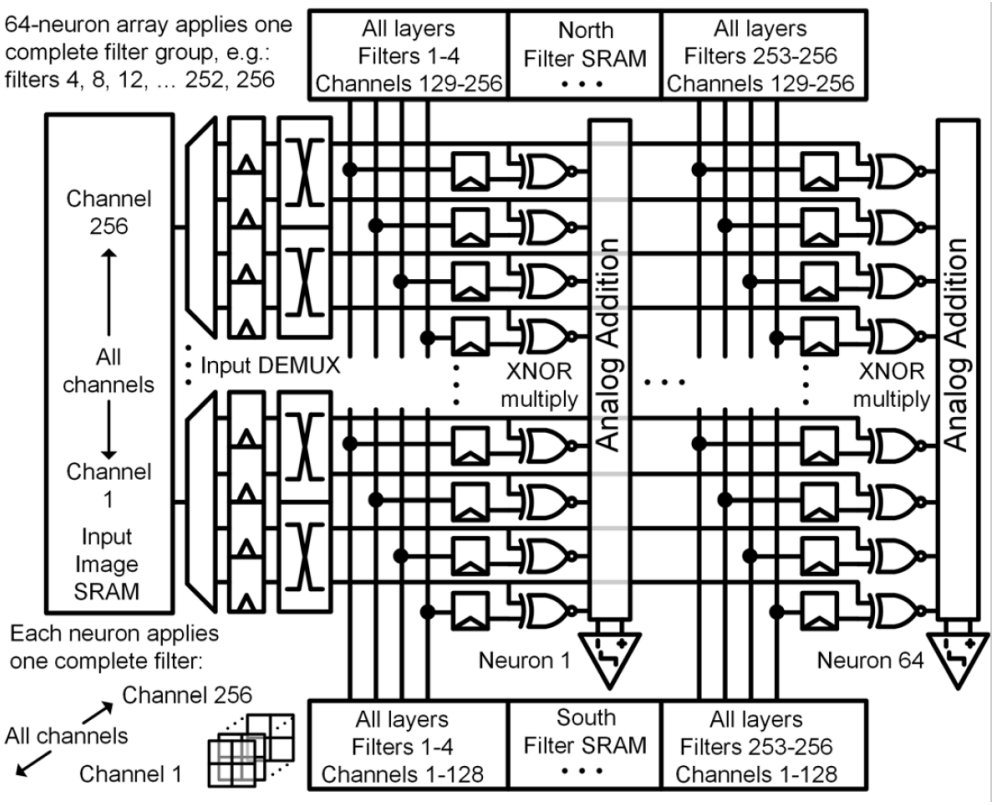


Figure 3.3: BNN architecture with SRAMs for synaptic weight storage and analog adders for neurons. Adapted from [6].

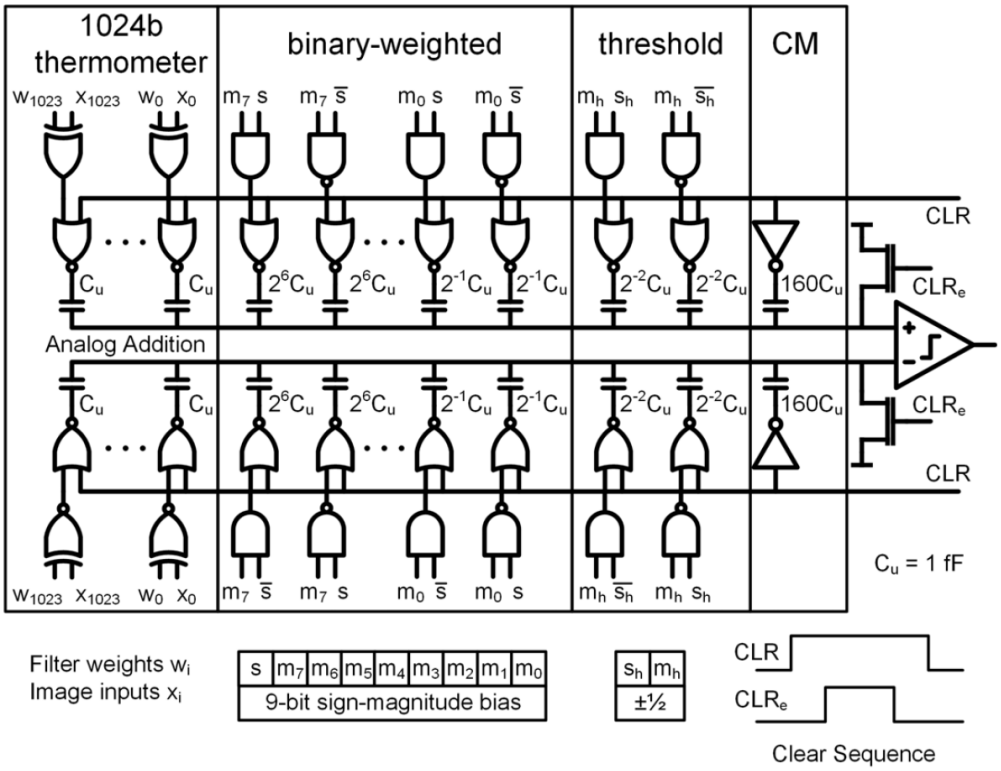


Figure 3.4: Switched-capacitor neuron using charge redistribution for wide vector summation. Adapted from [6].

final neuron outputs by accumulating the result of XOR gates. The key part of the architecture is the analog neuron made of switched-capacitors as shown in Figure 3.4. All the capacitors are discharged initially, and the active output of the XOR/XNOR gates charge the load capacitors. Then, the charges on the capacitors are equally redistributed and the input node of the comparator is driven by a stable voltage. By comparing the two input voltages, the comparator produces a neuron output. Due to the use of energy-efficient analog neurons, the fabricated chip implementing CNN on CIFAR-10 achieved 532 TOPS/W with 86.05% classification accuracy. However, the use of SRAM is a limiting factor in scalability.

In the BNN architecture of [7], ReRAM is used for on-chip weight storage. As

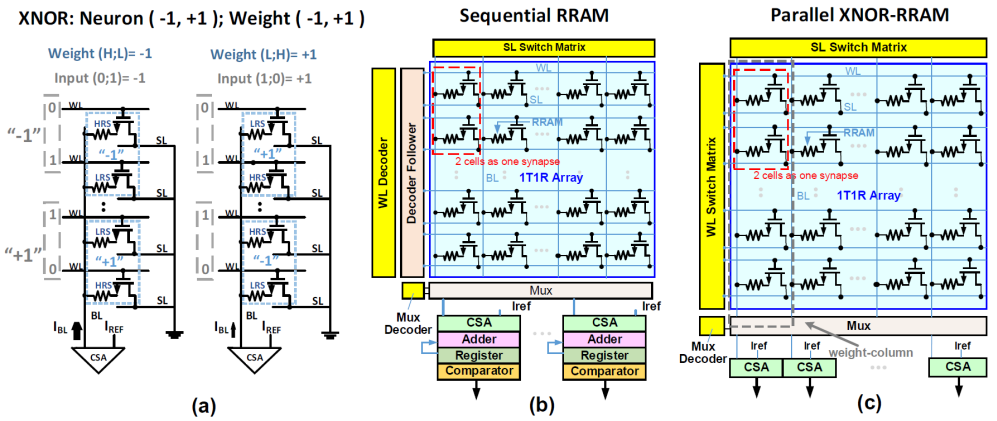


Figure 3.5: (a) The customized bit-cell design for XNOR implementation. (b) The diagram of conventional sequential RRAM synaptic architecture. (c) The diagram of proposed parallel XNOR-RRAM architecture. Adapted from [7].

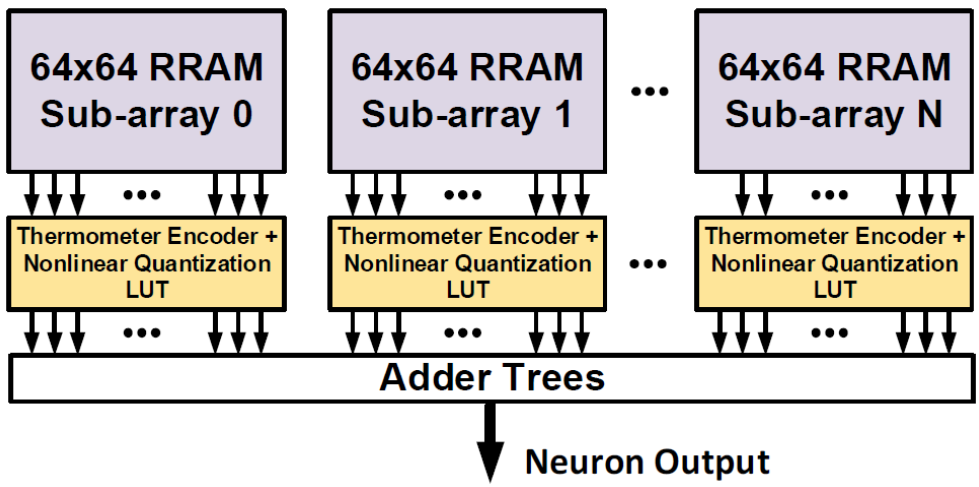


Figure 3.6: Generic system diagram for implementing one layer with arbitrary size in a network. Adapted from [7].

shown in Figure 3.5, a couple of ReRAM cells are used to represent a 1-bit weight. A set of ReRAM cell pairs drives an input of the following current sense amplifier (CSA), and it produces a multi-bit output value in a format of thermal code. The thermometer encoder and nonlinear quantization look-up table perform a compensation

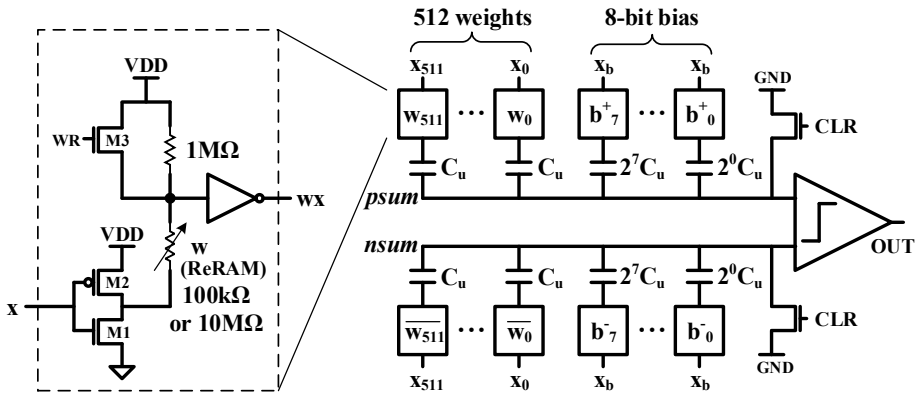


Figure 3.7: (a) a synapse circuit exploiting ReRAM for weight storage and (b) a neuron circuit using switched capacitors accompanying 512 weights and an 8-bit bias. C_u is set to 1fF. Adapted from [8].

against the nonlinearity caused by the ReRAM sub-array and produce partial sum output (Figure 3.6). Then, the digital adder tree accumulates all the partial sums and the sign of the accumulation result is generated as a neuron output. The experiment result of 256x256 matrix-vector-multiplication achieved 141 TOPS/W at 65nm technology. The use of ReRAM enables to keep all the weights in on-chip and avoid the use of off-chip memory which consumes enormous power. However, the use of digital logic units such as thermometer encoders, nonlinear LUTs, and adder trees offset the low power merit significantly.

The work in [8] proposed a BNN architecture with ReRAM synapses and switched-capacitor neurons. As shown in Figure 3.7, the ReRAM cell with a resistor acts as a voltage divider and produces near-0V or near-VDD output. The inverter in the next stage makes a full-swing output and drives the load capacitors, and the comparator compares the positive and negative partial sums and generates 1-bit neuron output. Thanks to the use of ReRAM synapses and analog neurons, the BNN architecture

achieved 1534 TOPS/W energy efficiency at 32nm technology.

Chapter 4

Deep Neural Networks with Weighted Spikes

4.1 SNN with weighted spikes

4.1.1 Weighted spikes

The main idea of the weighted spikes is assigning different weights to different phases (or to spikes in those phases) in order to pack more information into the spikes. This is the major difference from a conventional rate coding scheme that assigns the same weight to every spike. Figure 4.1 shows an example of weighted spike trains corresponding to the input values. A spike train consists of a sequence of periods, each of which has K phases of different weights. In our approach, we assign weights of $2^{-1}, 2^{-2}, \ldots, 2^{-K}$ respectively to the 1st, 2nd, $\ldots$, K-th phases, so that a period consists of phases divided by time, each of which corresponds to a different spike weight. Thus, spike weight $\omega(t)$ at current time $t \in \mathbb{N}^+$ (t is the total number of time steps elapsed from the 1st phase of the 1st period) is given by

$$\omega(t) = 2^{-(1+mod(t-1,K))}. \tag{4.1}$$

We normalize the values of all input signals and activations (ReLU outputs) to fit into the range $[0, 1 - 2^{-K}]$ by using the *data-based normalization* technique introduced in [36]. Then the weighted spike train within a period corresponds to a binary representation of the value with K fractional bits. We assume the use of a conventional frame-based sensor which produces multi-bit outputs. The multi-bit outputs of the sensor are fed to our SNN in a bit-by-bit manner from MSB to LSB. For example, consider an SNN for image classification task, where an input image sensor produces values corresponding to the intensities of 10 x 10 pixels. The most significant bits of all the 100 pixel outputs are fed to the SNN in time step 1 and then the second most significant bits are fed to the SNN in time step 2. This is continued to the end of the period at time step K. Then the same sequence is repeated for the following periods until the image is recognized. In the same way, a set of K-bit data is transferred between layers. This leads to significantly faster communication compared to SNN-RC

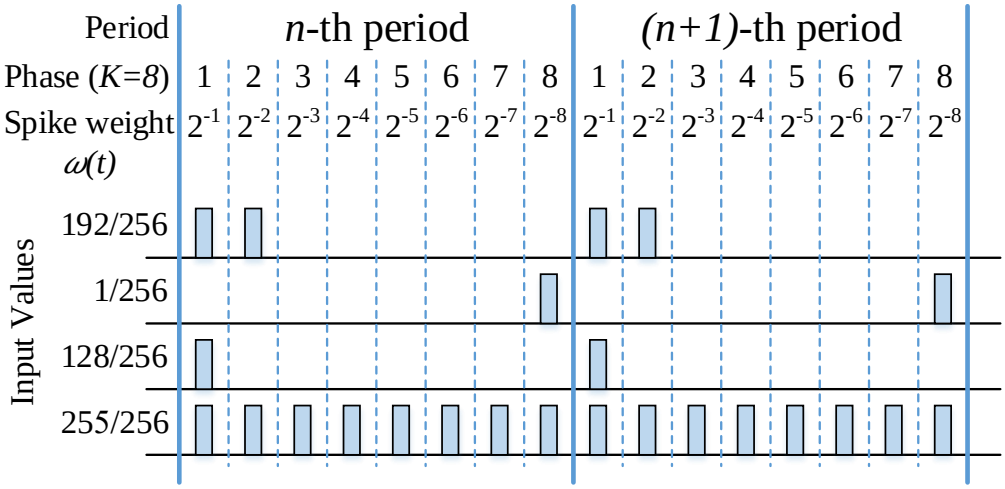


Figure 4.1: Example of input spike trains when the number of phases in a period (K) is 8. A rectangle indicates a spike, and each spike has the spike weight $\omega(t)$ depending on the phase of global reference clock. The same input spike trains are repeated until the end of classification.

since SNN-WS needs only K-slots to represent a K-bit data whereas SNN-RC would require 2^K-slots to represent the same data.

Due to the weighted spikes, SNN-WS can transmit more information with spikes than SNN-RC for the same time duration. It results in shortened classification latency and reduced number of spikes generated, which leads to higher throughput and energy-efficiency.

4.1.2 Spiking neuron model for weighted spikes

The adoption of weighted spikes requires modifications of equations for the conventional spiking neuron model described in Section 2.2. The modified spiking neuron model for weighted spikes is described in this section.

In ANN, the ReLU activation of the i-th neuron in layer l is given by

$$a_i^l = max\left(0, \sum_{j=1}^{M^{l-1}} W_{ij}^l a_j^{l-1} + b_i^l\right), \tag{4.2}$$

where M^{l-1} is the number of neurons in layer $l-1$, W_{ij}^l is the synaptic weight between layers $l-1$ and l, and b_i^l is the bias for the i-th neuron in layer l.

Similarly, in SNN-WS, the input current $z_i^l(t)$, which will be integrated into the membrane potential of the i-th neuron in layer l at time t—$z_i^l(t)$ corresponds to a bit of a_i^l in ANN—is represented as

$$z_i^l(t) = \omega(t)\left(\sum_{j=1}^{M^{l-1}} W_{ij}^l \tau s_j^{l-1}(t) + \beta_i^l\right), \tag{4.3}$$

where τ is the threshold of spiking neurons, $s_j^{l-1}(t)$ is the output spike of j-th neuron in layer $l-1$, and β_i^l is the normalized value of bias b_i^l in ANN and given by $\beta_i^l = b_i^l / \left(1 - 2^{-K}\right)$, which leads to $\sum_{t=1+(n-1)K}^{nK} \omega(t)\beta_i^l = b_i^l$ for $n \in \mathbb{N}^+$.

Equation (2.1) representing an occurrence of a spike still holds for SNN-WS. It can also be calculated by using the relation between an output spike and a membrane potential as

$$s_i^l(t) = U\left(V_i^l(t-1) + z_i^l(t) - \omega(t)\tau\right), \tag{4.4}$$

where $U(x)$ is a unit step function. $V_i^l(t)$ is the membrane potential of the i-th neuron in layer l, given by

$$V_i^l(t) = V_i^l(t-1) + z_i^l(t) - \omega(t)\tau s_i^l(t). \tag{4.5}$$

In summary, as shown in (4.3), $z_i^l(t)$ is obtained by first taking the sum of input spikes multiplied by the corresponding synaptic weights, adding the adjusted bias, and then scaling by $\omega(t)$. As shown in (4.4) and (4.5), the membrane potential is updated by integrating $z_i^l(t)$ (adding spikes to the membrane potential every time step). As shown in (4.4), the neuron fires an output spike with spike weight $\omega(t)$ if the updated

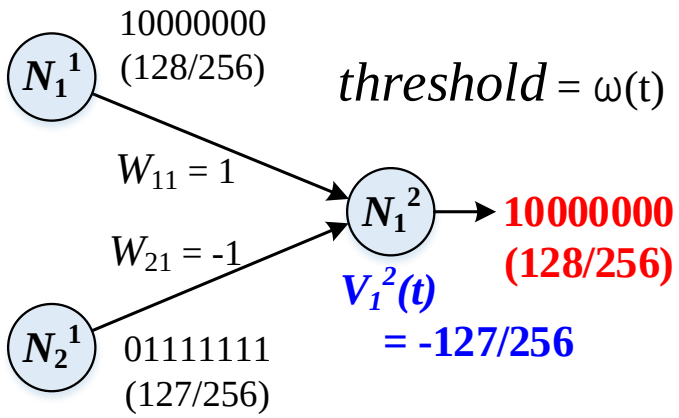


Figure 4.2: Example of a noise spike output. The neuron N_1^2 generates the noise spike train (10000000) instead of the correct one (00000001). The negatively charged membrane potential $V_1^2(t)$ prevents further generations of noise spike trains in next periods.

membrane potential is greater than or equal to the threshold scaled by $\omega(t)$. Finally, in (4.5), the membrane potential is subtracted by the threshold scaled by $\omega(t)$ if an output spike is generated (i.e., $s_i^l = 1$). In this work, we fix τ to 1 for all neurons and thus it is omitted in the rest of this paper for simplicity.

4.1.3 Noise spike

The bit-by-bit computation of weighted spikes in a spiking neuron can cause noise spikes[1] since a spiking neuron does not consider *a priori* information of the future input bits. Figure 4.2 shows a simple case of generating a noise spike when K is 8. In the figure, neuron N_1^1 generates a spike train 10000000 (128/256) and N_2^1 generates 01111111 (127/256) in the same period of 8 time steps. In the first time step, neuron N_1^2 receives a spike from N_1^1 and increases its membrane potential by 128/256 because

[1]The problem also exists in SNN-RC, but it's more significant in SNN-WS due to the higher information density.

the spike weight of the first phase is 2^{-1} and the synaptic weight is 1. Then, N_1^2 fires an output spike because its membrane potential 128/256 is sufficient to fire a 2^{-1} weighted spike. In the remaining time steps in the period, however, only N_2^1 outputs spikes, charging the membrane potential of N_1^2 in the negative direction and no longer producing any output spike. As a result, N_1^2 produces a spike train 10000000 and have membrane potential of -127/256, which is far from the correct output spike train 00000001 and the membrane potential 1/256.

Under this mechanism, unpredictable fluctuation of the output spike trains (noise spikes) generated during the first period makes it difficult to perform classification based on them. However, the negatively charged membrane potential $V_1^2(t)$ in the example above prevents additional noise spike generations in the following periods, which gradually compensates the error caused by the noise spike produced in the first time step. Therefore, the average of output spike trains over multiple periods can accurately approximate the desired value (we consider the ReLU activation of the corresponding neuron in ANN as the desired value). To compute the average of output spike trains, repetition of periods is required with the same input spikes until a certain level of classification accuracy can be reached. Nevertheless, the weighted spikes achieve the accuracy level much faster than the conventional rate coding since they deliver more information during the same time interval. Note that the weighted spike scheme requires only $O(logN)$ time steps to send one of N different values while the rate coding requires $O(N)$ time steps. This makes the SNN-WS achieve lossless accuracy with short classification latency and reduced number of spikes compared to the SNN-RC. In addition, we suggest to skip updating the output membrane potential for a few periods, so that the effect of noise spikes can be minimized and thus improve the classification accuracy and latency (see Section 4.2.1).

4.1.4 Approximation of the ReLU activation

In SNN-RC, average firing rate of a neuron approximates ReLU activation of the corresponding neuron in ANN [20]. In SNN-WS, the value of an output spike train averaged over periods approximates the corresponding ReLU activation more precisely. In this section, we describe how the averaged output spike train in SNN-WS correctly approximates the corresponding ReLU activation.

The signal value from the i-th input neuron (i.e., the i-th sensor output value) for the n-th period is given by

$$a_i^0 = \sum_{t=1+(n-1)K}^{nK} \omega(t) s_i^0(t), \tag{4.6}$$

for $n \in \mathbb{N}^+$. For the same input image, the signal value remains the same regardless of the value of n. In the i-th neuron of the first hidden layer, the following equations for the sum of weighted output spikes accumulated over n periods with $V_i^1(0) = 0$ can be derived from (4.5) and (4.3):

$$\begin{aligned}
\sum_{t=1}^{nK} \omega(t) s_i^1(t) &= \sum_{t=1}^{nK} z_i^1(t) - V_i^1(nK) \\
&= \sum_{t=1}^{nK} \omega(t) \left(\sum_{j=1}^{M^0} W_{ij}^1 s_j^0(t) + \beta_i^1 \right) - V_i^1(nK) \\
&= n \left(\sum_{j=1}^{M^0} W_{ij}^1 a_j^0 + b_i^1 \right) - V_i^1(nK) \\
&= n a_i^1 - V_i^1(nK),
\end{aligned} \tag{4.7}$$

If $a_i^1 \leq 0$, no output spike is generated, showing the behavior equivalent to ReLU. If $a_i^1 > 0$ and n is sufficiently large, $V_i^1(nK)$ becomes much smaller than na_i^1 (note that $V_i^1(nK) < \omega(t)\tau$) and can be ignored, which leads to

$$\frac{1}{n} \sum_{t=1}^{nK} \omega(t) s_i^1(t) \approx a_i^1. \tag{4.8}$$

In this manner, it can be generalized to other layers as follows:

$$\frac{1}{n}\sum_{t=1}^{nK}\omega(t)s_i^l(t) \approx a_i^l, \tag{4.9}$$

which means the average of the sum of weighted spikes over many periods correctly approximates the corresponding ReLU activation in ANN.

Therefore, the classification can be made by calculating

$$\arg\max_i \left(\sum_{t=1}^{nK}\omega(t)s_i^L(t)\right), \tag{4.10}$$

in the output layer L, which can be implemented with weighted spike counters at the end of the neurons in the output layer. However, instead of generating output spikes and counting them, we found it better to keep accumulating the membrane potential of each output neuron as follows:

$$V_i^L(nK) = \sum_{t=1}^{nK} z_i^L(t) \approx na_i^L. \tag{4.11}$$

Then the classification is made by computing

$$\arg\max_i \left(V_i^L(nK)\right). \tag{4.12}$$

It reduces the classification latency by saving the time to charge the membrane potentials of the output neurons to generate spikes.

We experimented with a spiking neural network (Net4 in Table 4.1) on CIFAR-10 dataset to verify the approximation capability of SNN-WS empirically. Figure 4.3 shows moving averages of the sum of weighted input spikes for 10 neurons in the output layer with a randomly selected CIFAR-10 test image. Each line converges to the value of its corresponding activation in ANN. Although the convergence requires around 200 time steps, the classification decision can be made much earlier (after 21 time steps) when the top membrane potential tends to maintain its top rank position. We also compared the approximation errors of the rate coding and the weighted spikes

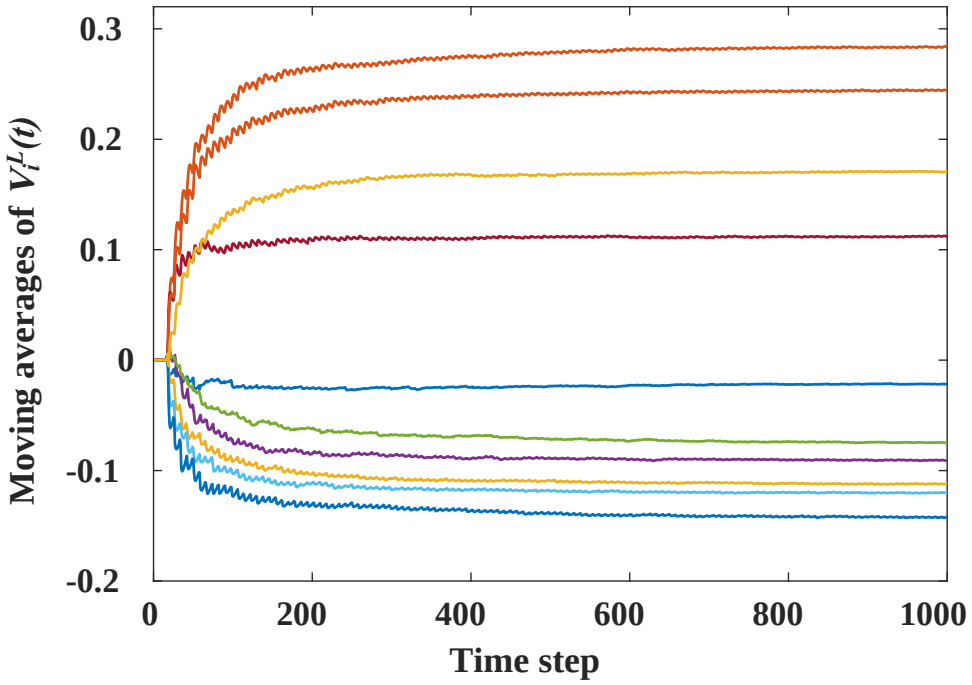


Figure 4.3: Moving averages of the membrane potentials for 10 neurons in the output layer. It is calculated from a randomly selected CIFAR-10 test image.

with Net4. The mean absolute errors (*MAE*) and the standard deviations (*STD*) of the approximation errors with SNN-RC and SNN-WS are depicted in Figure 4.4. It clearly shows that the approximation errors diminish over time, and SNN-WS converges faster to more accurate values.

4.1.5 ANN-to-SNN conversion

Since it is difficult to achieve high accuracy by training SNNs directly with spikes, it is a common practice to convert a trained ANN into an SNN. For the ANN-to-SNN conversion, we use the techniques introduced in [20] to reorganize biases and batch normalization layers [69]. To extend SNN-WS to convolutional neural networks (*CNN*s), we convert a max pooling layer of ANN by calculating the accumulated sum of weighted spikes for each neuron and then passing only the spike train that has the

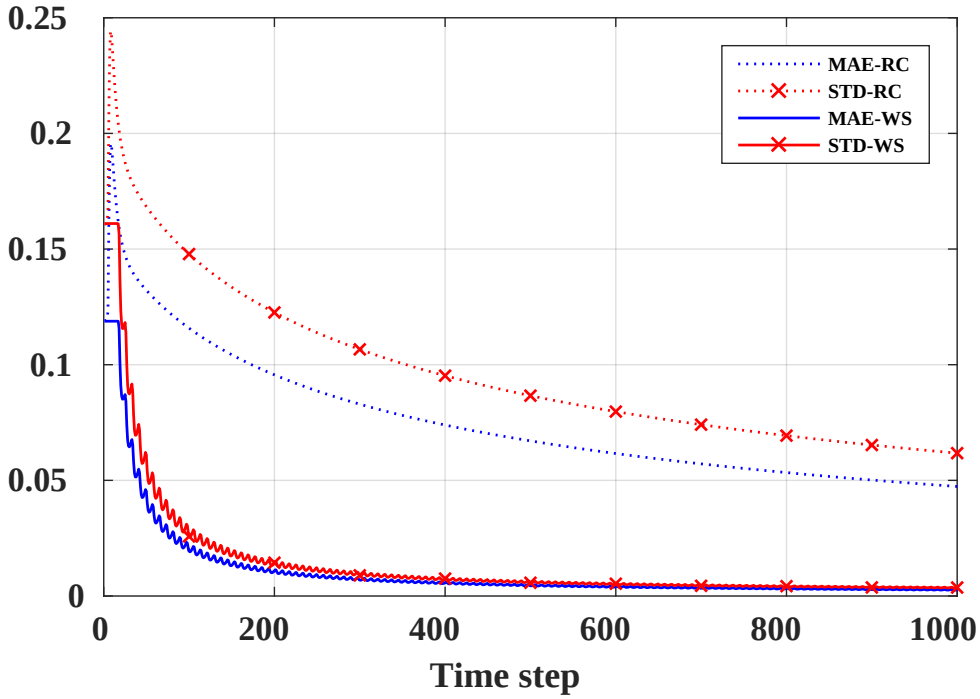


Figure 4.4: Mean absolute error (MAE) and standard deviation (STD) of the approximation errors for all activations in a spiking neural network using rate coding (RC) and weighted spikes (WS). It is calculated from the test results with 10,000 CIFAR-10 test images.

maximum accumulated sum. The softmax layer of ANN is not converted since the softmax layer is dispensable in an inference stage. Instead, the membrane potentials of the neurons in the output layer are directly compared to make a classification decision.

The previous approach [20] uses analog inputs rather than spike inputs to achieve higher accuracy and lower classification latency. However, the use of analog input requires different kinds of neurons in the first hidden layer, each of which must perform a lot of precise analog multiplications of input signals and synaptic weights. Thus, we do not use analog inputs even though the use of analog input leads to further classification speed up.

Deep residual networks [70] show superior classification accuracy than plain net-

works for various image recognition tasks. If a residual network is converted to an SNN without accuracy loss, the resulting SNN will achieve higher classification accuracy than an SNN converted from a plain network. In a residual network, there is a join point that merges the residual path and the shortcut path. The merge is done by adding the convolution output from the residual path with the output from the shortcut path. The method to convert this structure into an SNN is depicted in Figure 4.5. First the adders at the join point and its subsequent ReLU layer are replaced with new spiking neurons. Then the convolution weights of both residual and shortcut paths are moved to the synaptic weights of the new spiking neurons. The spiking neurons fed by the shortcut connection have twice as many synapses as the other spiking neurons.

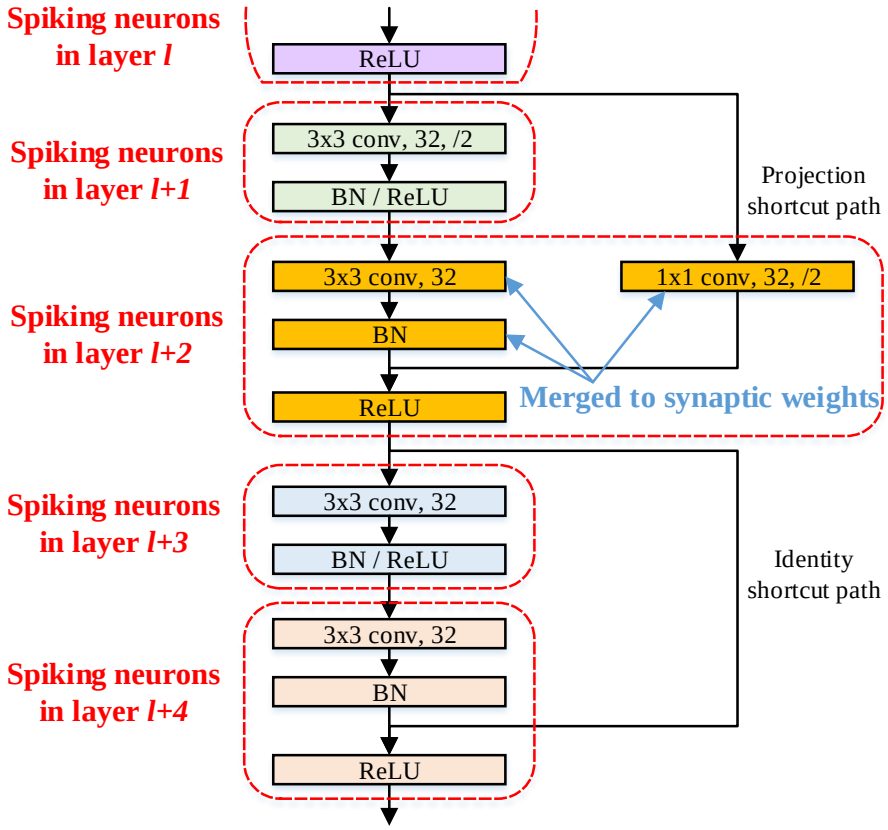


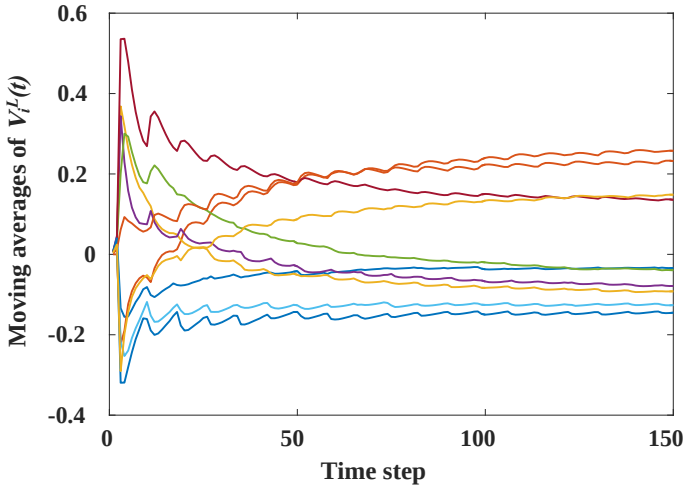
Figure 4.5: ANN-to-SNN conversion for a deep residual network. The convolutional and batch normalization layers on the residual and shortcut paths are merged with the following ReLU layer, and they form a population of spiking neurons in a layer.

4.2 Optimization techniques

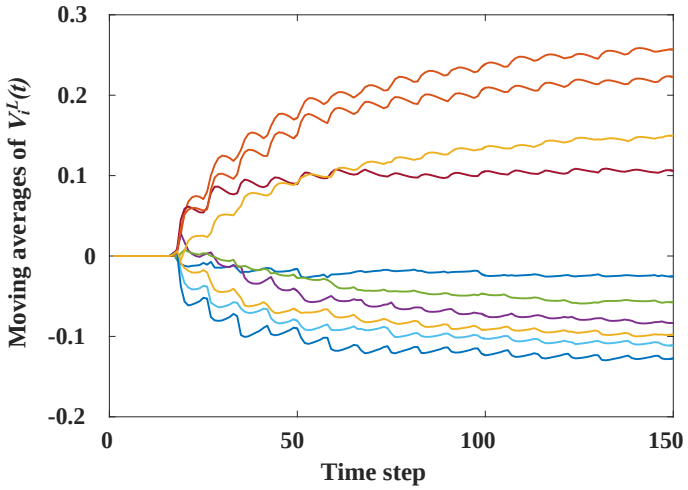
4.2.1 Skipping initial input currents in the output layer

To reduce the adverse effect of noise spikes, we introduce an input current skipping technique. The current skipping technique ignores the noisy input currents to the neurons in the output layer for a few initial periods, thereby improving the classification latency by a great amount. As explained in section 4.1.3, the highly-weighted bits of the input signals during these initial periods are propagated through the SNN without considering the following bits with lower weights and long term average. Moreover, during these periods, the membrane potentials of the neurons across the network are not initialized well. Thus, they may incur a lot of noise spikes and charge the membrane potentials of the spiking neurons in the output layer to an incorrect direction. Instead of accumulating the noisy input currents and then averaging out the effect in the following periods, we found it better to remove the effect just by ignoring the initial input currents and skipping the update of membrane potentials in the output neurons during the initial periods. This simple technique improves the classification latency significantly.

Figure 4.6 shows the effect of initial period skipping in the output layer. The results are obtained from an inference experiment with Net4 in Table 4.1 on CIFAR-10 dataset. When the skipping method is not applied (4.6a), the membrane potentials of the output neurons highly fluctuate due to the initial noise spikes and they rush into incorrect levels which are far from the correct converge points during the initial time steps. It takes long time steps to move the incorrectly initialized membrane potentials to the correct converge points. The classification decision is made by selecting the label of the output neuron which has the highest membrane potential. Thus, the correct decision can be made after 52 time steps. On the other hand, when the skipping method is applied (4.6b), the membrane potentials of output neurons can avoid interferences



(a) Do not skip the initial input currents in the output layer. The correct decision is made after 52 time steps.



(b) Skip the initial input currents in the output layer for 16 time steps. The correct decision is made after 21 time steps.

Figure 4.6: Effect of skipping initial input currents in the output layer. Each line shows the moving average of membrane potential for a neuron in the output layer. It is calculated from a randomly selected CIFAR-10 test image. Note that Y-axis scales are different.

caused by noise spikes during the initial 2 periods (16 time steps [2]), and move faster to the converge points. The correct decision can be made in only 5 time steps after the end of skipping the initial input currents, making the total delay to be 21.

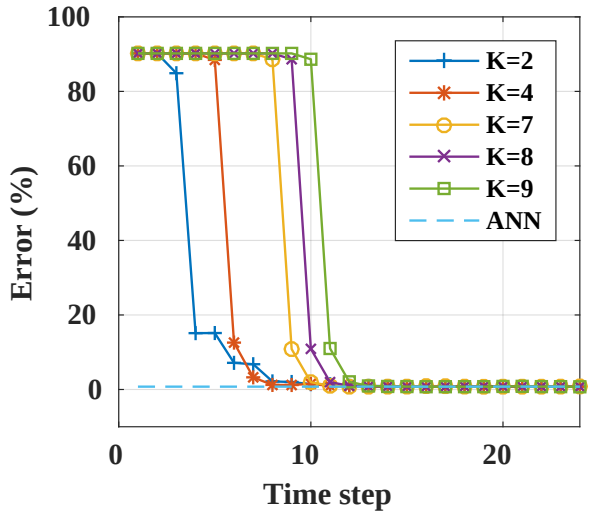
The optimal number of initial periods to skip charging in the output layer depends on the depth of the network. In general, it is better to skip more periods when the network is deeper since it takes longer time for input spikes to traverse a deeper network. In terms of noise reduction, it is actually better to skip as many periods as possible. However, excessive skipping leads to a too long classification latency, which is also undesirable. In order to achieve a reasonable accuracy and latency, the length of the skipping periods should be tuned empirically for each network depending on its depth and topology.

4.2.2 The number of phases in a period

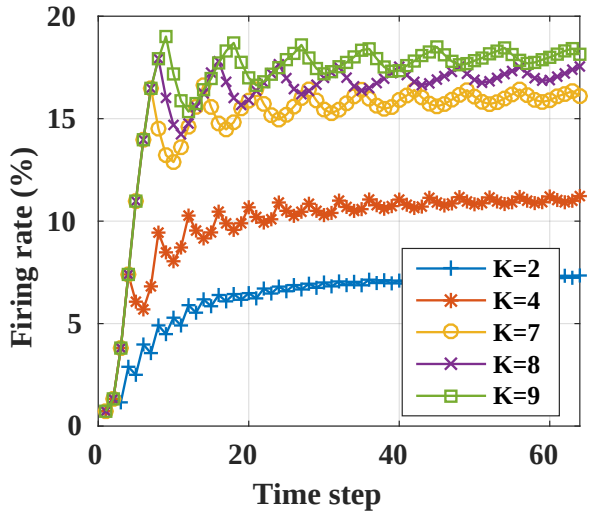
The number of phases in a period (K) is a tunable parameter affecting both latency and accuracy. By increasing the value of K, one can increase the classification accuracy at the cost of increased latency. In case of an image classification, a pixel intensity of an image is normally quantized to 8 bits for each of RGB channels. Therefore, setting $K = 8$ is usually enough to accurately deliver the pixel information to the neurons in the input layer. $K < 8$ causes a loss of the input signal precision, but it can reduce the classification latency. On the other hand, $K > 8$ incurs a waste of time because there will be no input spike for the neurons in the input layer during the phases over 8. However, the hidden layer can have a precision higher than 8 bits.

Figure 4.7a shows the classification error for various K parameters with Net2 in Table 4.1 on MNIST dataset. The membrane potentials of the output neurons are not updated for the first period in order to avoid initial noise spike. The time at which the classification error begins to fall is different depending on K because a period consists

[2] In this experiment, the number of phases in a period is set to 8. Thus, 1 period consists of 8 time steps.

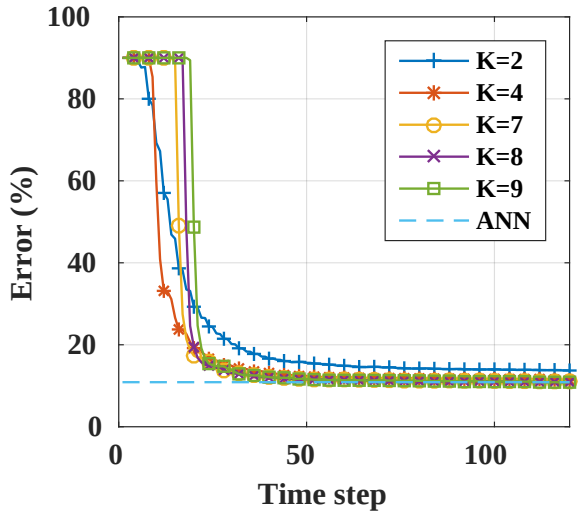


(a)

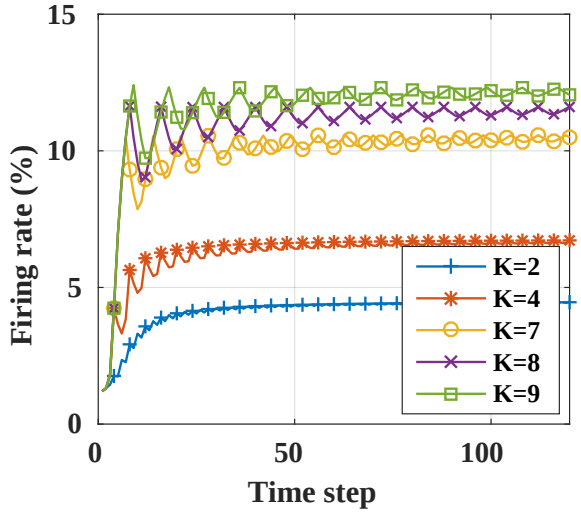


(b)

Figure 4.7: Comparison among different number of phases (K) in Net2 on MNIST dataset. $V_i^L(t)$ update is skipped for the first period. The smaller K leads to earlier start of the error convergence, but it results in worse converged accuracy. The average firing rate fluctuates within a period and the smaller K leads to lower firing rate. (a) Classification error over time. (c) Average firing rate over time.



(a)



(b)

Figure 4.8: Comparison among different number of phases (K) in Net4 on CIFAR-10 dataset. $V_i^L(t)$ update is skipped for the first 2 periods. It shows slower inference convergence when $K < 7$. (a) Classification error over time. (c) Average firing rate over time.

of K time steps. If we skip the update of membrane potentials of the output neurons for fixed time steps regardless of K, the input values fed into the SNN are distorted and it incurs accuracy loss. $K = 4$ results in the shortest classification latency on MNIST dataset under the assumption that 1%p accuracy loss is allowed. Even though the small K worsens the input signal precision, it does not result in a severe accuracy loss. It is since most pixels are biased toward either 0 (black) or 255 (white) in images of MNIST dataset, which makes the classification accuracy insensitive to the bits with lower weights.

Figure 4.8a shows the classification error with Net4 in Table 4.1 on CIFAR-10 dataset. In this case, smaller K leads to higher classification error at the end of convergence. Images on CIFAR-10 dataset use RGB color format and pixel value in each of the RGB channels is quantized to 8 bits. Unlike the MNIST dataset, pixel values of CIFAR-10 image are not biased toward either of the edges 0 and 255, but rather distributed broadly in the whole range [0, 255]. Thus, small K leading to omission of less significant bits incurs severe degradation of converged classification accuracy. According to our experiments, $K = 8$ yields the best performance on SVHN and CIFAR datasets.

Figure 4.7b and 4.8b compare the average firing rate over time among different K in Net2 and Net4. The average firing rate converges over time, but it shows a periodic oscillation within a period since near-0 activations are dominant in neural networks, which means the probability of generating a spike in a high-weight phase of a period is much lower than that in a low-weight phase in a period. Therefore, smaller K results in lower average firing rate.

4.2.3 Accuracy-energy trade-off by early decision

Among weighted spikes in a period, by transmitting the spike with the highest weight first, we allow the network to make a decision as early as possible (before the arrival of lower weight spikes) and thus terminate the classification earlier. For this

Algorithm 1 Early decision procedure:

Input: a time range $[T_1, T_2]$, membrane potentials of output neurons $\mathbf{V}^L(t) = \{V_1^L(t), V_2^L(t), \dots\}$ for $t \in [T_1, T_2]$, and stable decision threshold θ.

Output: class decision $D(t)$ and decision time t.

1: $D(T_1 - 1) \leftarrow \emptyset,\ \ \delta_{EMA}(T_1 - 1) \leftarrow 0,\ \ count \leftarrow 0$ // initialization
2: **for** $t \leftarrow T_1$ **to** T_2 **do**
3: $D(t) \leftarrow \arg\max_i V_i^L(t)$ // decision $D(t)$ in the current time step
4: $\delta(t) \leftarrow$ `top-2 difference of` $V_i^L(t)$ // δ represents classification difficulty
5: $\delta_{EMA}(t) \leftarrow (\delta(t) + \delta_{EMA}(t-1))/2$ // exponential moving average of δ
6: **if** $D(t) = D(t-1)$ **then**
7: $count \leftarrow count + 1$ // increase stable decision count
8: **if** $count \geq \theta/\delta_{EMA}(t)$ **then**
9: **return** // return if current decision has been stable for $\theta/\delta_{EMA}(t)$ time steps
10: **else**
11: $count \leftarrow 0$

purpose, we additionally define the "difficulty of classification" of an input image as the difference of top-2 activations among the output neurons. If the difference of the top-2 activations for an input image is small, it requires quite accurate computation to successfully distinguish the difference and make a correct decision. On the other hand, if the difference of top-2 activations is large enough, it can be distinguished even with relatively rough approximation. As mentioned in Section 4.1.4, the accuracy of SNN-WS increases over time, implying an accuracy-latency/energy trade-off.

Algorithm 1 demonstrates the procedure of early decision in this work. The early decision loop begins after T_1 in order to avoid the initial noise spikes. The current decision $D(t)$ is the class label of the output neuron that has the highest membrane potential (line 3). It computes the top-2 difference among the membrane potentials of

the output neurons (line 4), which represents the classification difficulty of the input image. It takes the exponential moving average as a stabilizer since the top-2 difference fluctuates over time (line 5). If the decision is unchanged for θ/δ_{EMA} time steps (line 8), the early decision is made and the classification finishes (line 9). The stable decision threshold θ contributes to an accurate early decision in spite of the fluctuations of the membrane potentials in the output layer. By tuning θ, we can take the trade-off between classification accuracy and energy saving which can be estimated by the reduced number of spikes. For a given value of θ, smaller δ_{EMA} (implying that the input image is more difficult to classify) requires larger value of $count$ and thus the decision takes longer to terminate.

4.2.4 Consideration on hardware implementation

The spike weight $\omega(t)$ is used for the calculations of the input current $z_i^l(t)$ and the output spike $s_i^l(t)$ as shown in equations (4.3) and (4.4). If SNN-WS is implemented with hardware strictly following those equations, every input spike and the threshold should be scaled by $\omega(t)$, which requires quite frequent scaling since the number of all input spikes in a network is enormous. It can be avoided by changing (4.4) to

$$s_i^l(t) = U\left(V_i^l(t-1)/\omega(t) + z_i^l(t)/\omega(t) - \tau\right). \tag{4.13}$$

Then, $\omega(t)$ in $z_i^l(t)$ is cancelled out, so that the input spikes and the threshold no longer require the scaling. Only the membrane potential needs to be scaled by $1/\omega(t)$, which can be easily implemented by a bit-shift operation.

4.3 Experimental setup

Experiments were performed with seven different networks on four different datasets: MNIST [59], SVHN [24], CIFAR-10, and CIFAR-100 [25].

MNIST It is a handwritten digit (0-9) image dataset containing 60,000 images for the training set and 10,000 images for the test set. Each image consists of 28 x 28 pixels, and each pixel is represented with 8-bit grayscale format.

SVHN It is a real-world image dataset obtained from house numbers (0-9) in Google Street View images. It contains 73,257 images for the training set, 26,032 images for the test set, and additional 531,131 extra images. Each image consists of 32 x 32 pixels, and each pixel is represented with 24-bit RGB format. We do not use the extra images in the experiments.

CIFAR-10 It is a real-world image dataset with 10 classes containing 50,000 images for the training set and 10,000 images for the test set. Each image consists of 32 x 32 pixels, and each pixel is represented with 24-bit RGB format.

CIFAR-100 It has the same image format as CIFAR-10, but it consists of 100 classes instead of 10. Each class has 500 training images and 100 test images.

Table 4.1 shows all the network configurations (Net1 through Net7) with ANN and SNN accuracies of our approach and SNN accuracies of previous approaches. Net1 is a multi-layer perceptron (MLP) including an input layer of 784 neurons, two hidden layers of 1200 neurons each, and an output layer of 10 neurons in order to verify that the proposed idea works in a typical neural network. Net2 is a typical CNN including two convolutional layers with 12 and 64 5x5 kernels (12c5, 64c5), two 2x2 max subsampling layers (2s, 2s), and a fully connected layer between the vectorized final features of the second subsampling layer and 10 output neurons. Those two networks

Table 4.1: Network configurations

Dataset	Net	Configuration	ANN Acc. (ours)	SNN Acc. (ours)	SNN Acc. (prev)
MNIST	Net1	MLP: 784-1200-1200-10 [36]	98.6%	98.6%	98.6%
	Net2	CNN: 12c5-2s-64c5-2s-10 [36]	99.2%	99.2%	99.1%
SVHN	Net3	NiN: 3 Mlpconv layers [71]	95.2%	95.2%	-
CIFAR-10	Net4	CNN: 32c3-32c3-2s-64c3-64c3 -2s-512-10 [20]	89.1%	89.2%	87.8%
	Net5	ResNet-20 [70]	91.4%	91.4%	-
CIFAR-100	Net6	ResNet-32 [70]	66.1%	66.2%	-
	Net7	Plain-32 [70]	64.3%	63.7%	-

are made with reference to [36], and their SNN accuracies are reported as 98.6% and 99.1% respectively.

Net3 is Network-in-Network [71] with three Mlpconv layers followed by a global average pooling layer. The first Mlpconv layer consists of 192 5x5 convolutional layer followed by two 192 1x1 convolutional layers. The second Mlpconv layer consists of 192 3x3 convolutional layer followed by two 192 1x1 convolutional layers. The third Mlpconv layer is the same as the second Mlpconv layer except that its last 1x1 convolutional layer has 10 feature maps instead of 192. The global average pooling layer takes the average of each feature map and produces 10 outputs. The Net3 is designed to have a large number of feature maps in order to verify that SNN-WS works in a wide neural network.

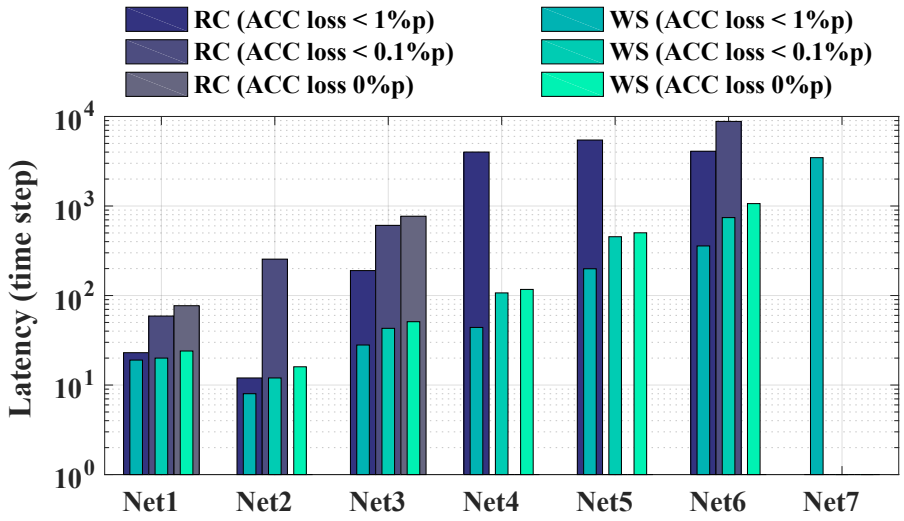
Net4 is a CNN with four 3x3 convolutional layers (2 x 32c3, 2 x 64c3), two 2x2 max subsampling layers (2 x 2s), and two fully connected layers originated from [20]. Its SNN accuracy is reported as 87.8% in [20]. Net5 and Net6 are residual networks

[70]. To the best of our knowledge, this is the first work to implement SNN with ResNet topology. Net5 consists of 19 convolutional layers (7 x 16c3, 6 x 32c3, 6 x 32c3) followed by a global average pooling layer and a fully connected layer producing 10 outputs. Net6 consists of 31 convolutional layers (11 x 16c3, 10 x 32c3, 10 x 32c3) followed by a global average pooling layer and a fully connected layer producing 100 outputs. There is a shortcut connection for each of two convolutional layers as shown in Figure 4.5. Net7 is a plain network obtained by removing the shortcut connections from Net6, which is added to check to see if the residual network shows faster inference convergence compared to the plain network.

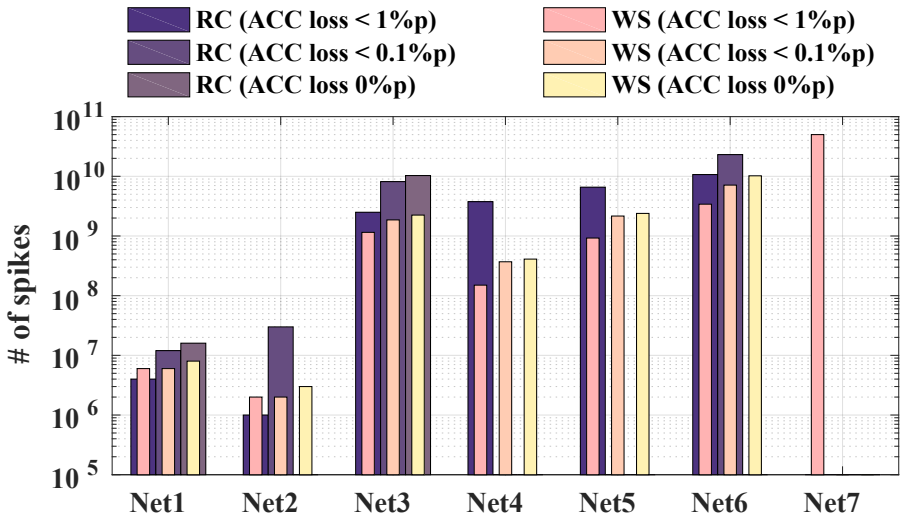
The spiking neural networks cannot accept negative input signals, thus the image preprocessing techniques producing negative values such as subtracting the per-pixel mean [72] and whitening [73] cannot be used. Instead, we normalized the input values into the range $[0, 1 - 2^{-K}]$, which causes zero or marginal accuracy loss compared to the training results with the image preprocessing techniques. The batch normalization [69] is used to train all the networks. For classifiers learning CIFAR dataset, the simple data augmentation with 4-pixel zero padding followed by random crop and random horizontal flip [74] is applied to the training images. The networks are trained with Keras [75] and MatConvNet [76]. The inferences of the networks were performed on MatConvNet.

4.4 Results

4.4.1 Comparison between SNN-RC and SNN-WS



(a) Classification latency



(b) Number of spikes

Figure 4.9: Experimental results for the networks. Some bars are omitted if they do not reach the target accuracy within 10,000 time steps. Note that the Y-axis is plotted on a log scale.

The experimental results with the seven networks on the four different datasets are summarized in Figure 4.9. It compares the classification latency and number of spikes between SNN-RC and SNN-WS. The experiments were performed with three different stop conditions depending on the allowed accuracy loss from the original ANN accuracy. In all the networks, SNN-WS shows much shorter classification latency compared to SNN-RC regardless of the network type, and the number of spikes is proportional to the latency. Note that the Y-axis of each chart is log scale.

For SNN-WS on MNIST dataset, the number of phases (K) in a period is set to 4, which means only the upper 4 bits of input pixel values are used in the inference. SNN-WS does not incur any accuracy loss in spite of the dropping of lower 4 bits, whereas SNN-RC cannot reach the no-accuracy-loss point within 10,000 time steps in Net2. We set $K = 8$ for other datasets because reducing K causes rather longer classification latency and accuracy loss.

Net3 shows a relatively large number of spikes because it is a wide network with 192 feature maps for each convolution layer. When the network depth grows, SNN-RC suffers from the significant increase of the classification latency, which limits the use of SNN-RC in practice. In Net5 and Net6, SNN-RC does not reach no-accuracy-loss point in 10,000 time steps while SNN-WS reaches there in 502 and 1064 time steps, respectively even though SNN-WS also suffers from the superlinear increase of latency on the increase of network depth. In SNN-RC, Net5 takes more time steps than Net6, even though Net6 is deeper than Net5. It is because Net5 has higher ANN accuracy which requires more time to increase the accuracy. In SNN-WS, on the other hand, Net6 takes more steps than Net5 because it is deeper. The accuracy of SNN-WS makes the classification latency less sensitive to the level of ANN accuracy.

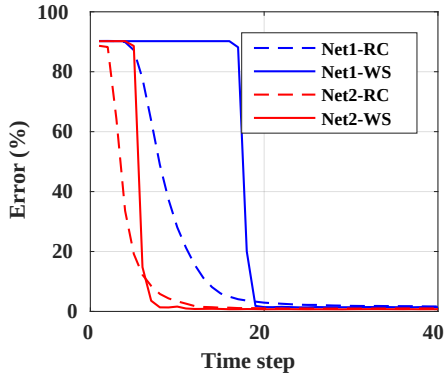
Interestingly, the residual network Net6 shows shorter classification latency compared to its plain counterpart Net7 that does not have any shortcut connection. The shortcut path in the residual network is intended to train deep layers better, but it also helps to boost the inference speed of SNN by propagating spikes faster to the deep

Table 4.2: Reduction of latency and # of spikes by SNN-WS compared to SNN-RC

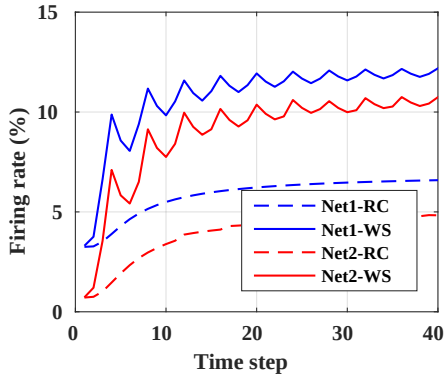
	Latency reduction			# Spike reduction		
ACC loss	<1%p	<0.1%p	0%p	<1%p	<0.1%p	0%p
Net1	1.2x	3.0x	3.2x	0.7x	2.0x	2.0x
Net2	1.5x	21.3x	-	0.5x	15.0x	-
Net3	6.8x	14.0x	15.1x	2.2x	4.4x	4.6x
Net4	91.1x	-	-	25.1x	-	-
Net5	17.2x	-	-	4.4x	-	-
Net6	11.4x	11.9x	-	3.1x	3.2x	-
Net7	-	-	-	-	-	-

layers through the shortcut path.

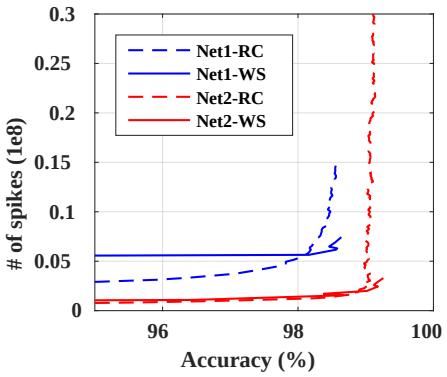
Table 4.2 represents the improvements of classification latency and number of spikes achieved by SNN-WS compared to SNN-RC. SNN-WS shows better performance in terms of latency and number of spikes except for the cases where 1%p accuracy loss is allowed on MNIST dataset. In a typical CNN such as Net2 and Net4, SNN-WS achieves significantly better performance than SNN-RC. The maximum latency reduction is 91.1x and the maximum spike reduction is 25.1x in the experiments of Net4 with 1%p accuracy loss allowed. The results of some conditions cannot be compared because SNN-RC cannot reach the target accuracy within 10,000 time steps. For the example of Net7, SNN-RC reaches 2.8%p accuracy loss point in 10,000 time steps while SNN-WS reaches 1%p accuracy loss point in 3,469 time steps.



(a)

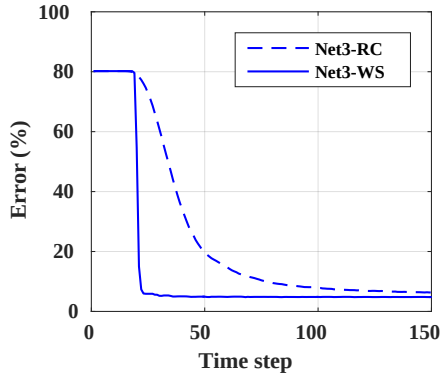


(b)

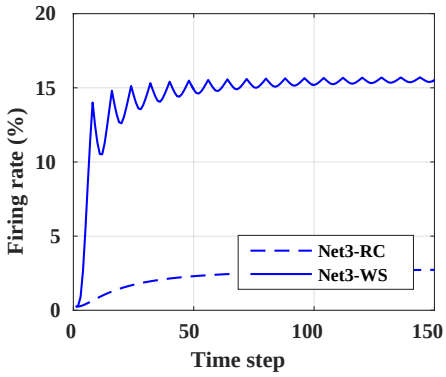


(c)

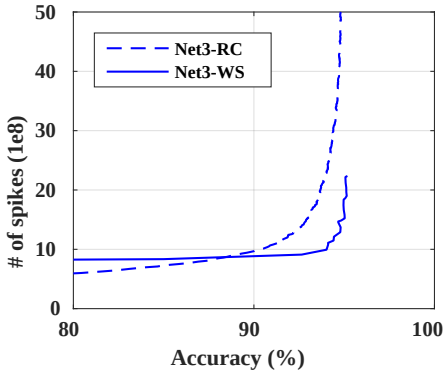
Figure 4.10: Experimental results (classification error over time, average firing rate over time, and number of generated spikes over accuracy) of Net1 and Net2. The number of phases (K) in a period is set to 4. $V_i^L(t)$ updates are skipped for 4 and 1 periods, respectively.



(a)

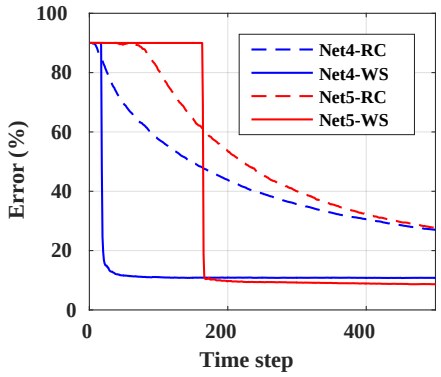


(b)

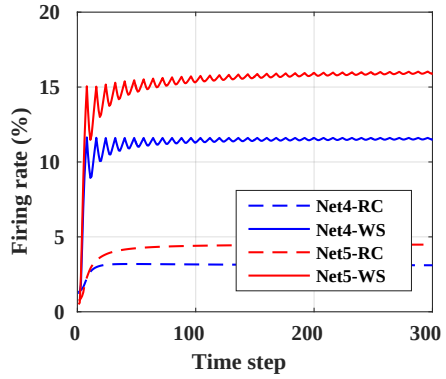


(c)

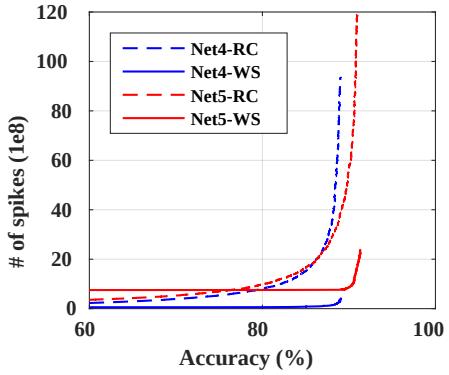
Figure 4.11: Experimental results (classification error over time, average firing rate over time, and number of generated spikes over accuracy) of Net3. The number of phases (K) in a period is set to 8. $V_i^L(t)$ updates are skipped for 2 periods.



(a)

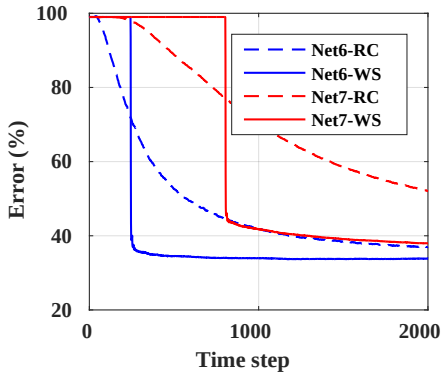


(b)

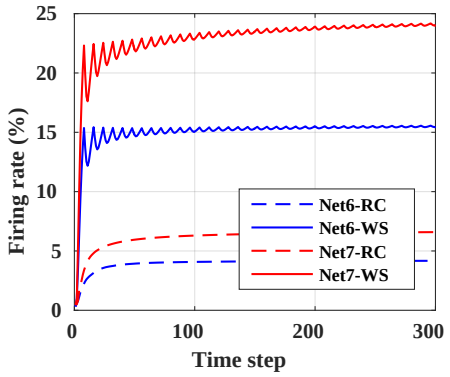


(c)

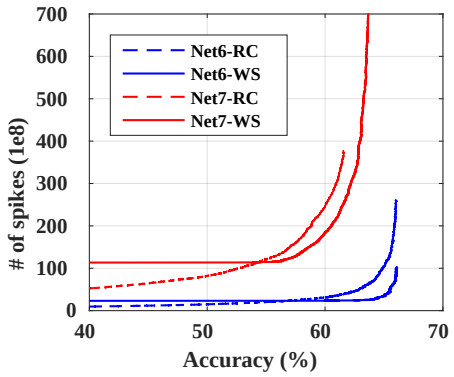
Figure 4.12: Experimental results (classification error over time, average firing rate over time, and number of generated spikes over accuracy) of Net4 and Net5. The number of phases (K) in a period is set to 8. $V_i^L(t)$ updates are skipped for 2 and 20 periods, respectively.



(a)



(b)



(c)

Figure 4.13: Experimental results (classification error over time, average firing rate over time, and number of generated spikes over accuracy) of Net6 and Net7. The number of phases (K) in a period is set to 8. $V_i^L(t)$ updates are skipped for 30 and 100 periods, respectively.

Figure 4.10, 4.11, 4.12, and 4.13 show the detailed experimental results of Net1 through Net7. The changes of classification error over time are shown in Figure 4.10a, 4.11a, 4.12a, and 4.13a. SNN-WS shows significantly faster inference convergence than SNN-RC after skipping the membrane potential updates of output neurons for initial periods. The skipping makes the SNN-WS reach no-accuracy-loss point much faster than SNN-RC although SNN-WS starts the inference convergence later than SNN-RC. SNN-RC shows comparable classification latency compared to SNN-WS in a shallow network (Net1 and Net2), but the classification latency of SNN-RC can not be compared to that of SNN-WS in deeper networks (Net3 through Net7). In very deep networks such as Net5, Net6, and Net7, the update of membrane potentials in the output layer should be skipped for many periods since it takes long time for the initial noise spikes to reach the output layer.

Figure 4.10b, 4.11b, 4.12b, and 4.13b plot the average firing rates over time. The average firing rates of both SNN-RC and SNN-WS converge over time. But the firing rate of the SNN-WS oscillates within a period as explained in Section 4.2.2. The converged firing rate of SNN-WS is 2$\sim$4 times larger than that of SNN-RC. It is because, for the dominant near-0 activations, the spikes in SNN-WS are more densely packed within a period than those in SNN-RC. For example, if a neuron generates an output activation value of 3/256 in ANN, the corresponding neuron in SNN-RC fires spikes at the rate of 3/256 because the firing rate of SNN-RC is proportional to the activation value of ANN. On the other hand, the corresponding neuron in SNN-WS fires spikes for 2 phases out of 8 phases which means a firing rate of 2/8.

Figure 4.10c, 4.11c, 4.12c, and 4.13c show the required number of spikes to achieve a specific accuracy. Even though the average firing rate of SNN-WS is larger than that of SNN-RC, the total number of generated spikes of SNN-WS is smaller due to the significantly shorter classification latency. In SNNs, the number of spikes is proportional to the dynamic energy consumption, which means that SNN-WS consumes much lower energy than SNN-RC.

4.4.2 Trade-off by early decision

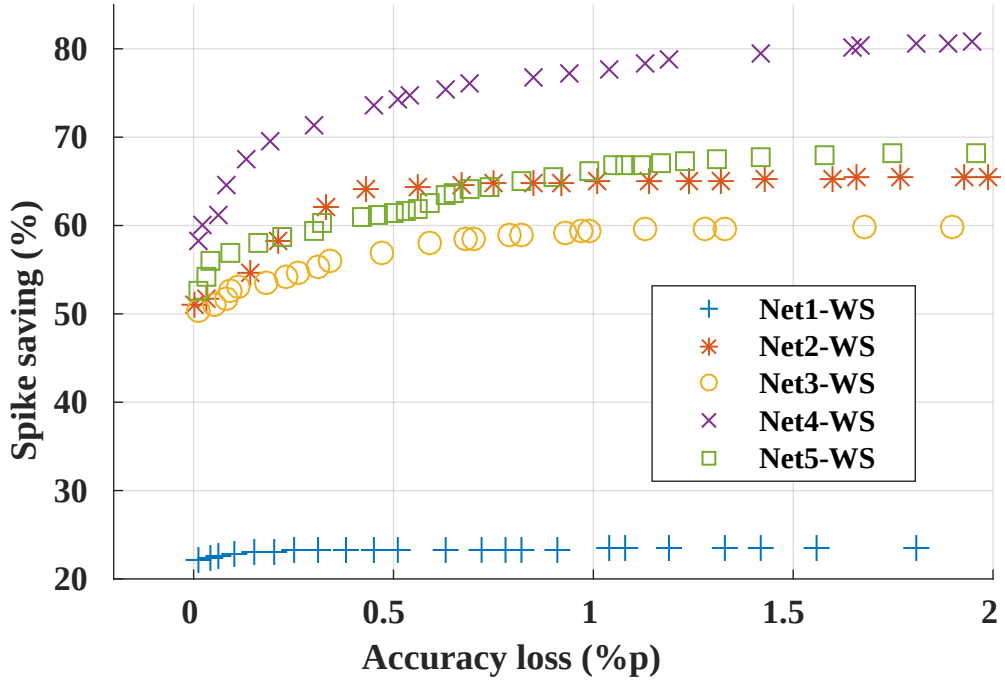


Figure 4.14: Trade-off between accuracy and number of spikes by the early decision algorithm.

Another advantage of SNN-WS is the early decision method that gives a chance for further energy reduction at the expense of small accuracy loss. Figure 4.14 shows the trade-off between accuracy and energy when the early decision method is exploited. Net2 through Net5 achieve 50%∼58% spike savings without accuracy loss, and 59%∼77% spike savings within 1%p accuracy loss. Such huge savings are possible since most test images show relatively large differences of top-2 activations compared to the approximation errors, which lead to shorter classification latency by early decision. Net1 represents relatively low spike savings (22%∼23%). In Net1, the earli-

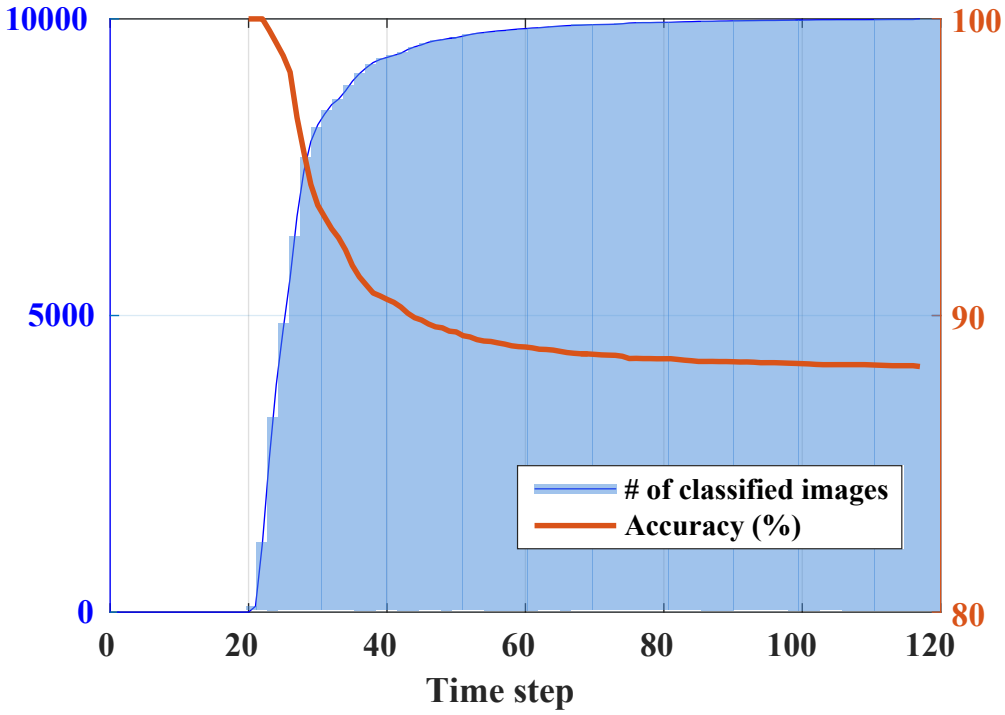


Figure 4.15: Number of classified images (left) and classification accuracy (right) over time when the early decision scheme is applied in Net4 with 1%p allowed accuracy loss. The stable decision threshold (θ) is set to 0.9.

est decision takes 17 time steps [3] and the latest decision takes 24 time steps. The small difference of the required time steps to classify between the least and the most difficult images diminishes the room of further improvement.

Figure 4.15 shows the number of classified images and accuracy achieved until the corresponding time step when the early decision scheme is exploited in Net4 with 1%p allowed accuracy loss. The number of classified images increases steeply after 21 time steps (classification decision cannot be made during the initial 2 periods because the membrane potentials of output neurons are not updated for the initial 2 periods) and about 94% of test images are classified with 90.55% accuracy within only 40

[3] In Net1, a classification decision cannot be made for the initial 16 time steps because K is set to 4 and the membrane potential updates of output neurons are skipped for the initial 4 periods.

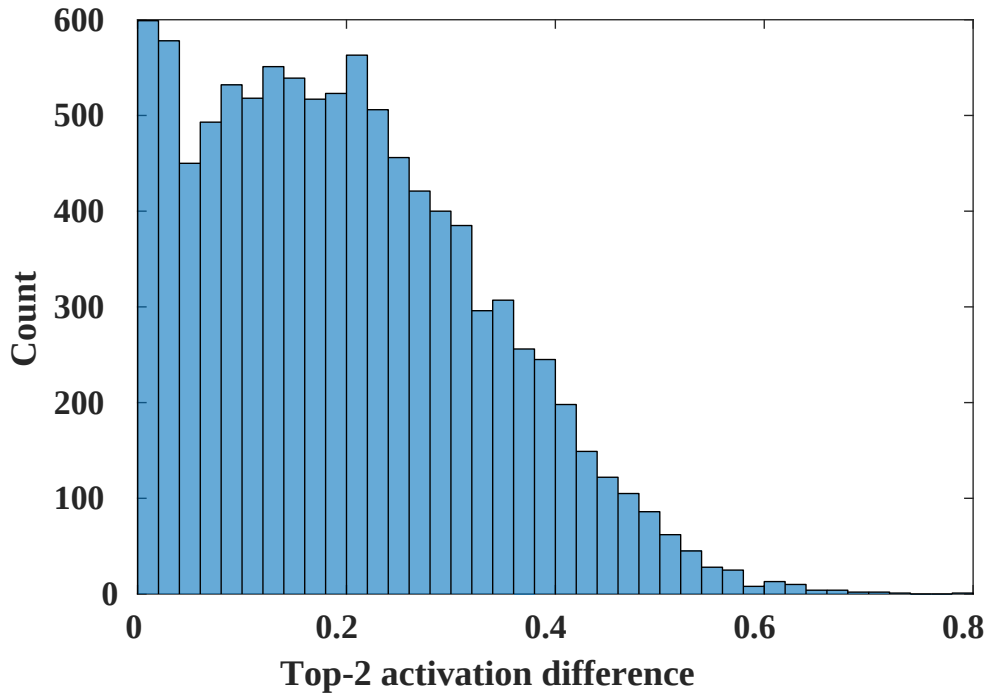


Figure 4.16: Histogram of top-2 ReLU activation difference in the output layer of Net4 (ANN version) with 10,000 CIFAR-10 test images.

time steps. The remaining 6% images, most of which belong to the leftmost bar in Figure 4.16, consume more time steps since they are difficult to classify correctly due to the small top-2 activation differences of output neurons. The last image (most difficult to classify) is classified in 117 time steps and the final accuracy of SNN-WS with early decision is 88.16% which is 0.94%p lower than the ANN accuracy (89.1%) of Net4. By the early decision scheme, the average classification latency is reduced from 117 time steps down to 27.6 time steps, which leads to 77% spike saving.

4.4.3 Comparison with other algorithms

Table 4.3: Comparison with other SNN algorithms on Iris dataset

SNN algorithm	Spike coding	Network	Acc.	# Neurons	# Params	Latency	# Spikes
SpikeProp [47]	Temporal	50-10-3	96.1%	63	24000	10	-
NPBLR [57]	Temporal	48-4-3	95%	55	60	400	-
Dyn. threshold [42]	Temporal	9-42-3	96.0%	54	168	340	-
ROL [58]	Relative order	48-3	94.2%	51	144	50	-

Table 4.4: Comparison with other SNN algorithms on MNIST dataset

SNN algorithm	Spike coding	Network	Acc.	# Neurons	# Params	Latency	# Spikes
ROL [58]	Relative order	784-10	90.3%	884	50000	100	-
Spike-DBN [60]	Rate	MLP	94.1%	1794	0.6M	-	-
STDP [44]	Rate	Two layers	95.0%	7184	5.0M	350	-
Weight norm. [36]	Rate	MLP (Net1)	98.6%	3194	2.4M	20	1×10^6
SNN-WS w/o ED[4]	Weighted spike	MLP (Net1)	98.6%	3194	2.4M	24	8×10^6
SNN-WS w/ ED	Weighted spike	MLP (Net1)	98.5%	3194	2.4M	19	6×10^6
Weight norm. [36]	Rate	CNN (Net2)	99.1%	24784	29816	80	1×10^6
SNN-WS w/o ED	Weighted spike	CNN (Net2)	99.2%	24784	29816	16	3×10^6
SNN-WS w/ ED	Weighted spike	CNN (Net2)	99.1%	24784	29816	8	1×10^6

[4]ED means the early decision algorithm described in Section 4.2.3. 0.1%p accuracy loss is allowed here.

Table 4.5: Comparison with other SNN algorithms on CIFAR-10 dataset

SNN algorithm	Spike coding	Network	Acc.	# Neurons	# Params	Latency	# Spikes
Spiking CNN [62]	Rate	CNN	77.4%	38090	0.1M	400	2×10^7
99.9% norm. [20]	Rate	CNN (Net4)	87.8%	118282	2.2M	280	-
SNN-WS w/o ED	Weighted spike	CNN (Net4)	89.2%	118282	2.2M	117	4×10^8
SNN-WS w/ ED	Weighted spike	CNN (Net4)	89.1%	118282	2.2M	42	1×10^8

Table 4.3, 4.4, and 4.5 show the comparison results of spike coding, classification accuracy, network size, classification latency in time steps, and number of generated spikes among different SNN algorithms on Iris, MNIST, and CIFAR-10 datasets. In the SNN algorithms based on temporal coding [42, 47, 57], information is packed in a spike timing, and it has higher information capacity than a rate coding and the proposed weighted spike. Thus they can perform classification tasks with less number of spikes theoretically. However, due to the difficulty of training based on spike timing, they are applied on a small dataset such as Iris (4 input features and 3 classes) [77], and there is no experimental result on larger dataset such as MNIST or CIFAR. To be applied on larger datasets, SNNs using temporal coding still require further researches.

The work on relative order learning [58], which encodes information on a relative order between spikes, shows that it is feasible on Iris and MNIST datasets. However, its classification accuracy on MNIST is just 90.3% which is far from the state-of-the-art accuracy 99%, and the method does not work on networks with more than two layers. Rate coding is used in most SNN algorithms [20, 36, 44, 60, 62] that show high classification accuracy on MNIST and CIFAR-10. The proposed SNN-WS achieves shorter classification latency than those with rate coding.

The number of spikes generated in a network is important since it is proportional to dynamic energy consumption. In SNNs, an incoming spike changes the membrane potential of a neuron and the process consumes dynamic energy and no dynamic energy is consumed if there is no spike. The SNN algorithms with temporal coding and

relative order coding do not give the number of generated spikes since they are mainly focused on achieving high accuracy and training with deep and large networks.

In the work using rate coding [36], the number of generated spikes is less than or equal to that of SNN-WS. Since SNN-WS has higher average firing rate than SNN-RC, SNN-WS is not much competitive compared to SNN-RC when a small network is used because SNN-WS has higher average firing rate than SNN-RC and a small network does not have much room to reduce classification latency further by using SNN-WS. However, SNN-WS shows significant reduction in number of spikes for large networks as shown in Table 4.2. Although the work using rate coding [20] does not give the number of generated spikes for the case of Net4 on CIFAR-10[5], our experimental result of SNN-RC with Net4 on CIFAR-10 shows that more than 4×10^9 spikes are generated during an inference, which is much larger than that of SNN-WS. Moreover, exploiting the early decision algorithm enhances the classification latency and number of spikes further and makes SNN-WS faster and more energy-efficient.

[5]We set the network configurations of Net1 and Net2 with reference to [36] and that of Net4 to [20] for comparison.

4.5 Summary

We proposed a novel spiking neural network with weighted spikes to overcome the slow classification problem of the conventional SNNs with rate coding. The key idea is assigning different weights to spikes depending on the time phase within a period to encode more information within less number of spikes. The proposed SNN-WS reduces the classification latency as well as number of spikes significantly. The approach can be applied to various types of networks including deep residual networks and various datasets without accuracy loss. We also proposed an early decision algorithm to further reduce latency and number of spikes at the expense of small accuracy loss.

Chapter 5

VCAM: Variation Compensation through Activation Matching for Analog Binarized Neural Networks

5.1 Modification of Binarized Neural Network

5.1.1 Binarized Neural Network

The binarized neural network (BNN) is a neural network using binary weights (-1 and +1) and activations (-1 and +1) [2]. BNN significantly reduces memory size and memory accesses due to the binary weights and activations. It also replaces multiplication operations to bit-level logical operations, which reduces power and area of processing units.

BNN uses the following sign function as an activation function.

$$x^b = Sign(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{otherwise,} \end{cases} \tag{5.1}$$

where x^b is the binarized weight or activation value from the real value x. However, since the derivative of the activation function is zero (except at $x = 0$), it is incompatible with the backpropagation algorithm. To cope with the problem, the authors in [2] use the following straight-through estimator to calculate gradients in the backward computation.

$$g(x) = \begin{cases} x, & \text{if } |x| \leq 1 \\ 0, & \text{otherwise,} \end{cases} \tag{5.2}$$

where $g(x)$ is the gradient function that makes BNN trainable with the backpropagation algorithm.

5.1.2 Use of 0 and 1 Activations

We modify the original BNN algorithm to use 0 and 1 for activations instead of -1 and +1 (we still use -1 and +1 for weights). In the original BNN, the use of -1 and 1 for weights and activations simplifies conventional multiply-and-accumulate (MAC) operation to XOR operation between weights and activations, and bit counting operation

among the XOR results. However, if we use 0 and 1 for activations, each XOR operation can be replaced with an AND operation which can be implemented with a single transistor in our synapse circuit design. Thus, it contributes to a more energy-efficient hardware implementation.

The activation function is slightly modified to produce 0 and 1 outputs as

$$x^b = f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5.3}$$

The approximated gradient function in (5.2) remains unchanged since it is applicable to both original and modified activation functions. According to our experiments, this modification of BNN algorithm incurs zero or marginal accuracy loss.

5.1.3 Removal of Batch Normalization Layer

Batch normalization (BN) layers [78] play an important role in BNN training. BNN training does not work properly without the BN layers. The BN function is as follows:

$$BN(x) = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}(x - \mu) + \beta, \tag{5.4}$$

where x is an output of the previous layer, μ is a running mean, σ^2 is a running variance, γ and β are learning parameters, and ϵ is a constant added to the running variance for numerical stability. The variables $\mu, \sigma, \gamma,$ and β are floating point numbers, thus they require expensive floating point adders and multipliers.

To avoid the hardware overhead of implementing BN layers, we merge those float-

ing point variables into a new variable α as follows:

$$
\begin{aligned}
f(BN(x)) &= f\left(\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}(x-\mu)+\beta\right) \\
&= \begin{cases} f\left(x-\mu+\frac{\beta\sqrt{\sigma^2+\epsilon}}{\gamma}\right), & \text{if } \gamma > 0 \\ f\left(-(x-\mu+\frac{\beta\sqrt{\sigma^2+\epsilon}}{\gamma})\right), & \text{otherwise} \end{cases} \\
&= \begin{cases} f(x+\alpha), & \text{if } \gamma > 0 \\ f(-(x+\alpha)), & \text{otherwise.} \end{cases}
\end{aligned} \tag{5.5}
$$

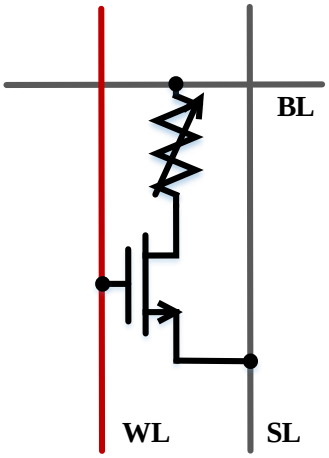
We add the constraint $\gamma \geq 0.5$[1] during the weight update stage of backpropagation to avoid the occurrence of output inversion. Then the BN layers can be removed by merging α onto the bias of the previous layer in the inference stage (BN layers are still required in the training stage).

Equation (5.5) holds for the intermediate BN layers followed by activation layers. In other words, it does not hold for the last BN layer since there is no activation layer after the last BN layer. In the last BN layer, since the activation function $f()$ is not used, merging the scale factor $\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$ into the term α gives a different result. However, for a low-cost implementation, we keep applying the same technique even for the last BN layer by constraining all the scale factors in the last BN layer to have the same value. By applying these two constraints to BN layers, we can completely remove the BN layers in the inference stage, and it does not incur accuracy loss compared to the original BNN algorithm.

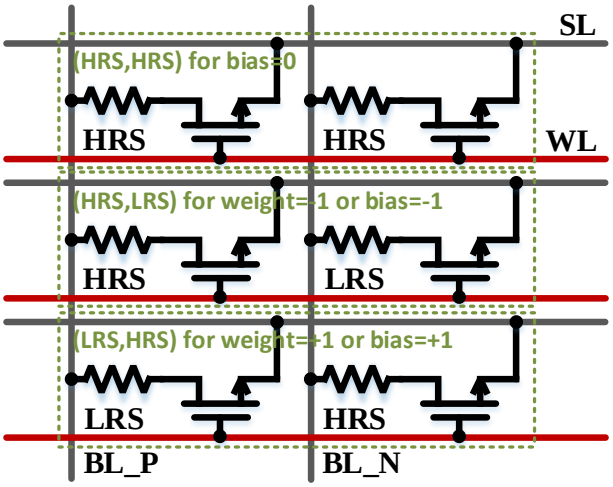
[1]The value 0.5 is selected empirically.

5.2 Hardware Architecture

5.2.1 ReRAM Synaptic Array



(a)



(b)

Figure 5.1: (a) Single ReRAM cell. (b) Two ReRAM cells which represents a binary weight or a fraction of bias.

The cell of ReRAM we adopt is composed of one resistor followed by one transistor, which is called 1T1R memristor. In contrast to the conventional memristor crossbar array (MCA) where a cell is just composed of one resistor, 1T1R MCA isolates each resistor from adjacent cells to solve the sneak-path problem [29] which aggravates the accuracy of output signals and energy efficiency.

Figure 5.1a shows the structure of the 1T1R memristor which can be programmed into two states: high-resistive state (HRS) and low-resistive state (LRS). R_{HRS} is set to $50\,\mathrm{M}\Omega$ and R_{LRS} to $2\,\mathrm{M}\Omega$. In write mode, the state of the memristor changes from HRS into LRS by applying V_{set} to the bit-line (BL), GND to the select-line (SL), and VDD to the word-line (WL). Similarly, its state changes from LRS into HRS by applying V_{reset} to the SL, GND to the BL, and VDD to the WL. In the read mode, current flows through BL according to the programmed value of resistor when applying GND to the SL and VDD to the WL [79]. V_{set} and V_{reset} can be varied according to the type of the memristor or system environment [80].

We use a pair of ReRAM cells to represent a binary weight (+1/-1) or a fraction of bias (+1/0/-1) as shown in Figure 5.1b. When the cell connected to BL_P is in HRS and the other one connected to BL_N is in LRS, the pair represents -1 for the weight or bias. Conversely, when the cell on BL_P is in LRS and the other is in HRS, the value is equal to +1. Since the bias of a neuron can be set to an arbitrary value including 0, multiple bits (or multiple pairs of cells) are assigned and the value of each pair can be set to 0, +1, or -1. Setting the value of a pair to 0 can be done by setting both cells to HRS.

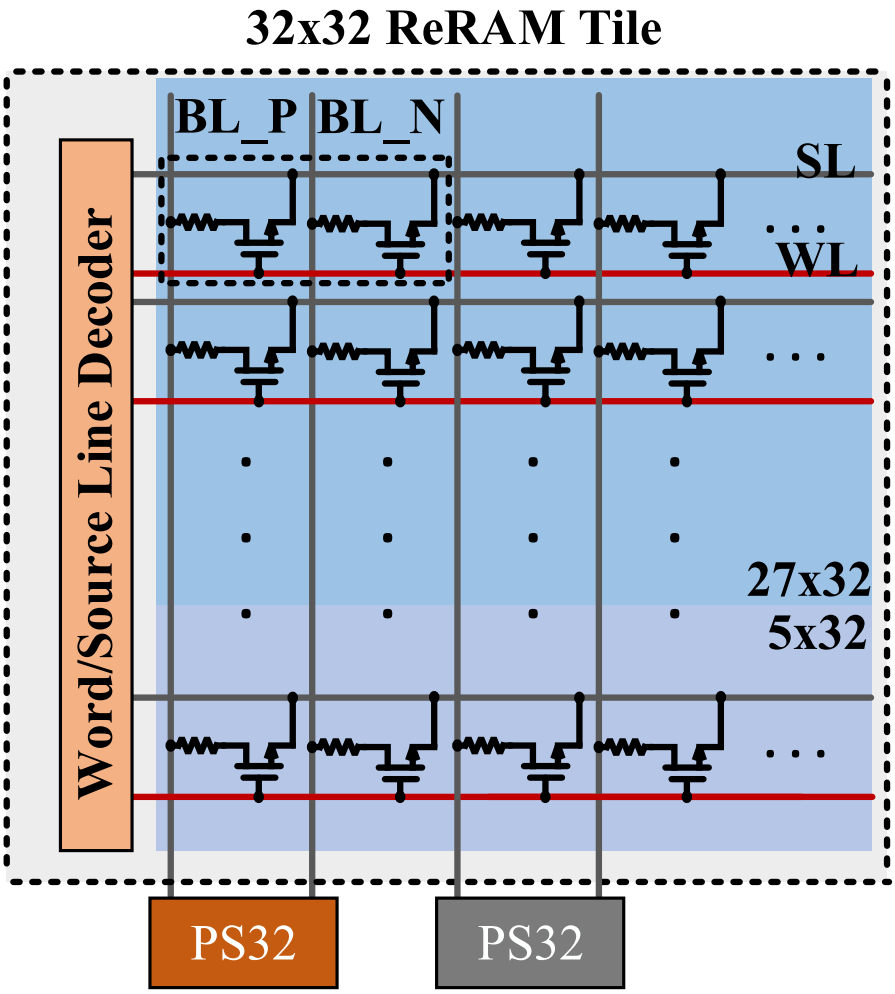


Figure 5.2: A ReRAM tile.

Figure 5.2 shows the 1T1R ReRAM tile. We set the size of a tile of ReRAM as 32 by 32, considering the property of ReRAM and PartialSum32 circuit block (PS32). Every two adjacent columns represents part of binary weight vector and fractions of bias. In a column pair, 27 paired cells are allocated for weights and 5 cells for bias. The proportion of bias cells can vary for each layer. Output signals of the previous

layer are propagated through the word/source line decoder[2], and thus WLs are set to high (+1) or low (0). In this way, the transistor of ReRAM cell can be used as both a sneak-path isolator and an AND gate that is controlled by WL which contains output activations of the previous layer.

After the training stage, pre-trained weights and biases are programmed on the ReRAM, where they are grouped and spread out. In the inference stage, all the SLs in all tiles are connected to GND and WLs are selected according to the output activations of the previous layer. The current which represents positive partially weighted sum flows through BL_P and the current which represents negative partially weighted sum flows through BL_N.

[2]During inference, the decoder connects all input signals to word lines in parallel.

5.2.2 Neuron Circuit

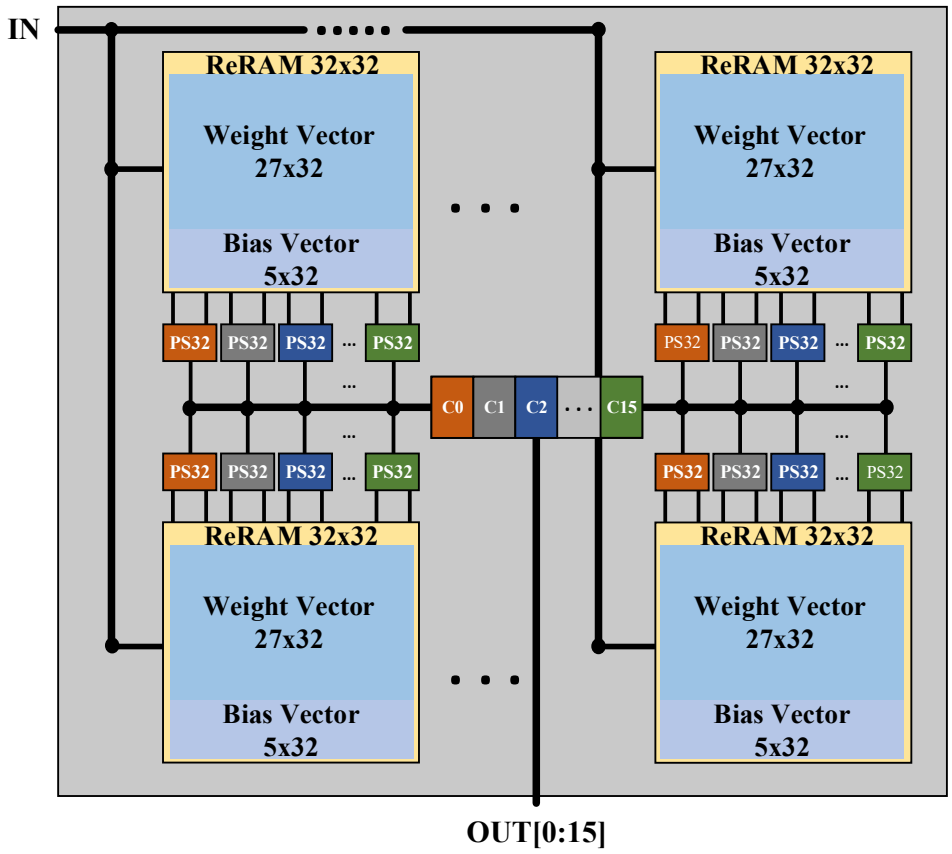


Figure 5.3: A neuron sub-array.

A neuron circuit is composed of ReRAM tiles, PS32s, and a comparator as shown in Figure 5.3. Due to ReRAM tiles arrangement, 16 neurons are grouped and compose one neuron sub-array. Input activation signals (IN) are transferred to ReRAM tiles. The ReRAM tiles perform AND-operations between weights and input activation signals, and then produce positive or negative partial sums, each of which is a summation of the 32 AND-operation outputs. PS32 subtracts the negative partial sum from the positive partial sum ,and then produces the result as V_{POUT}. After all, the comparator generates

a binary output by comparing the average of output voltages (V_{POUT}) of PS32s and the reference voltage (V_{ref}).

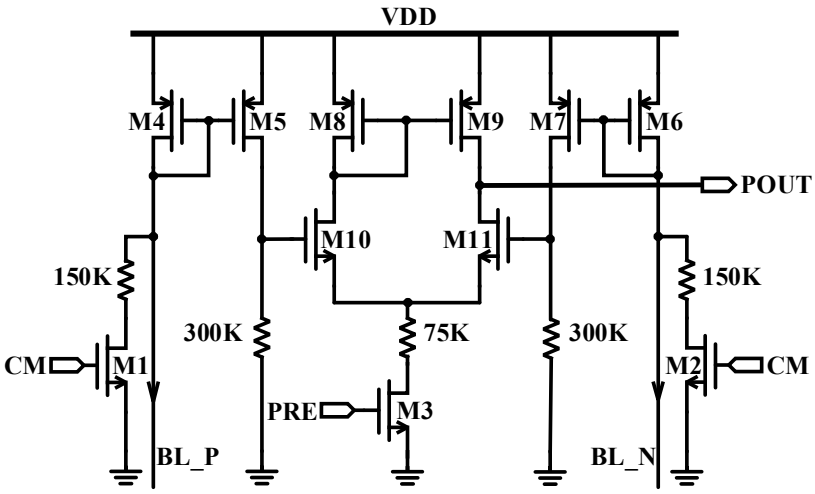


Figure 5.4: A PartialSum32 (PS32) circuit.

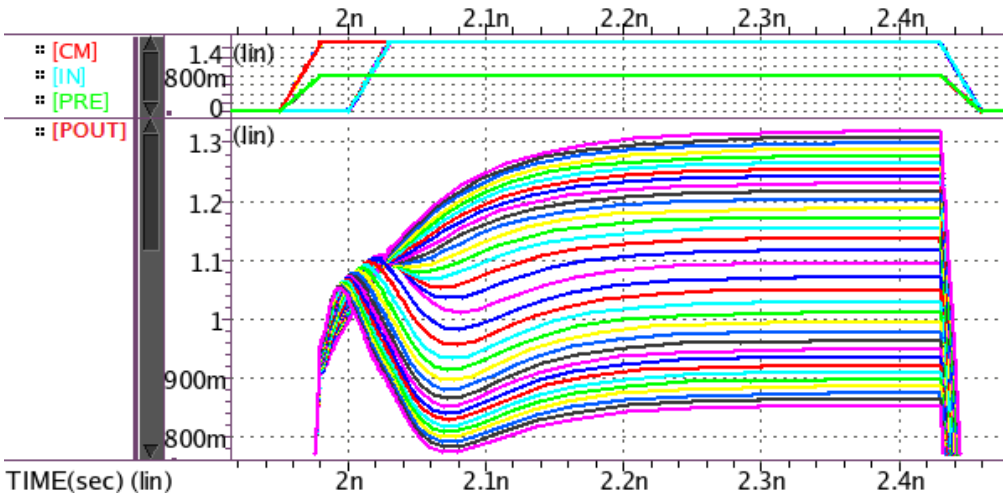


Figure 5.5: SPICE transient analysis result of PS32 for various partial sum inputs in 32nm technology. (up) control signals (CM, PRE) and input activation signal. (down) voltage at POUT node.

Figure 5.5 shows input control signals and SPICE transient analysis result of PS32 (Figure 5.4) for various partial sum inputs in a range from +32 to -32. First, the control signals CM and PRE are set before the input signals through WLs are activated in order to make the DA operate in linear region. Second, input activation signals are transmitted to the word-line decoder and then WLs are set to high (+1) or low (0). Through each corresponding ReRAM cell, overall currents are summed up on BL_P or BL_N which represent partially positive or negative weighted sum. Third, the currents are mirrored and scaled down to save energy consumption by M5 and M7. Mirrored currents are transformed into input voltages of DA by the followed resistor of M5 and M7. Then, stabilized out signal of PS32 is generated within 0.4ns after subtracting from partially positive weighted sum to partially negative weighted sum through the DA.

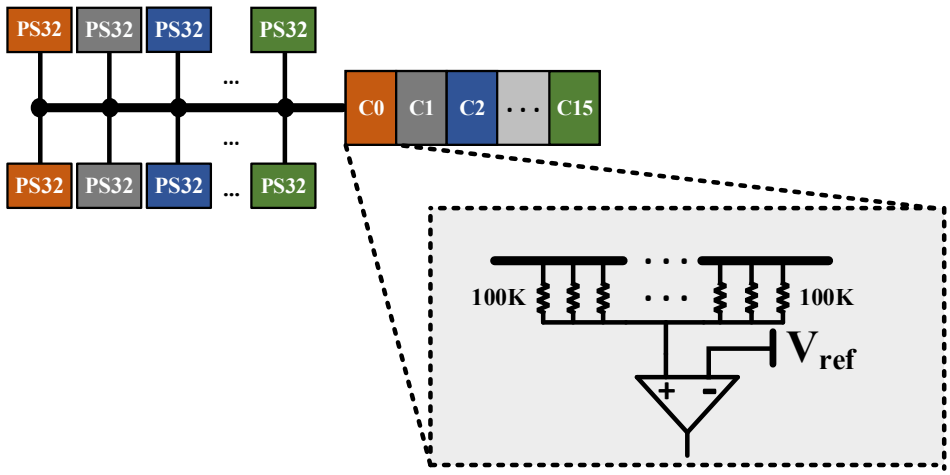


Figure 5.6: A comparator circuit.

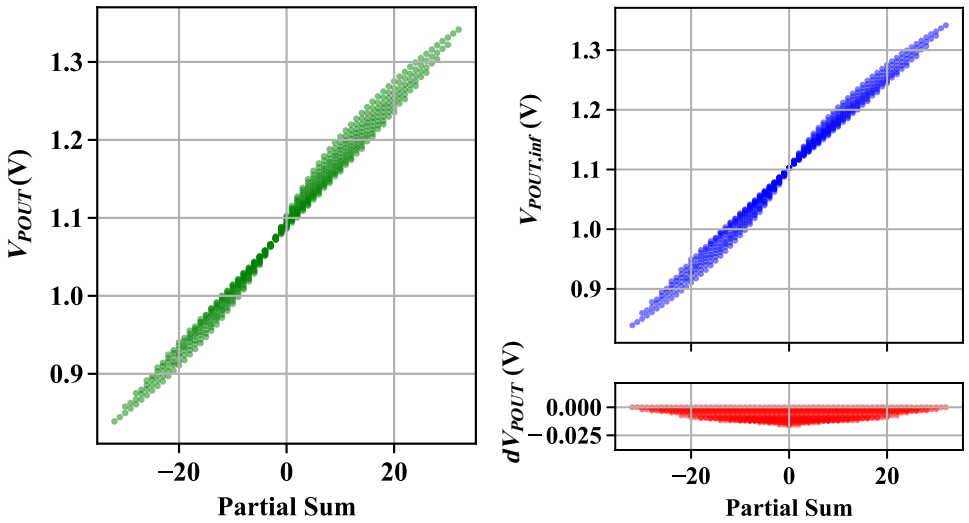
All output signals of PS32s are averaged on the positive input node of the comparator as shown in Figure 5.6. The negative input node of the comparator is connected to external reference voltage source which generates V_{ref} stably. V_{ref} is initially determined with the voltage when the weighted sum of a neuron is 0. By comparing the

voltages, the output activation signal of neuron is produced as: high (+1) if the average of partial sums is greater than V_{ref}, low (0) otherwise.

5.2.3 Issues with Neuron Circuit

BNN is good to be implemented with analog circuits because it is robust to noise [81]. However, as the neural network becomes bigger, several other issues arise. A common problem with the increase of fan-in is that the weighted sum spans over a wider voltage range with greater non-linearity, thus leading to an incorrect decision at the output [30]. To cope with these problems, input synapses are clustered into multiple smaller groups, and then each group sums up its own input activation signals to generate a partial sum. In [7], 64 input synapses are grouped and the partial sum values (originally ranging over 128 levels) are further quantized into 8 levels (3 bits) by multiple-level sensing amplifiers (MLSAs) composed of 1-bit current sense amplifiers (CSAs). However, the offset of CSAs caused by process variation worsens sensing accuracy at around the reference value of CSAs. This effect leads to test accuracy drop. In addition, the digital implementation of the adder trees decreases energy efficiency.

To enhance energy efficiency, we implemented the adder trees in analog (PS32) as introduced in Section 5.2.2. Each PS32 covers a group of 32 input synapses. Figure 5.7a shows the output voltage V_{POUT} of PS32 for all possible input combinations. V_{POUT} indicates partial sum converted to voltages. Note that different input combinations for the same partial sum can produce different output voltages due to the non-linearity of the analog circuit across overall combinations. In addition, finite common mode rejection ratio (CMRR) of the DA in PS32 further increases the variance of V_{POUT}. There are 17 different input combinations of positively weighted partial sum (PWPS) and negatively weighted partial sum (NWPS) that sum up to 0; every combination of (PWPS, NWPS) from (0, 0) to (16, 16) corresponds to 0 partial sum but the output voltage can be different due to the finite CMRR. This effect on V_{POUT} is represented as dV_{POUT} in Figure 5.7b. dV_{POUT} decreases as the sum of PWPS and



(a) (b)

Figure 5.7: Output voltages of (a) PS32, and (b) decomposed into two components by CMRR effect.

NWPS increases because the output voltage of our DA decreases as the input common mode voltage increases. Thus, the output voltage can be represented by

$$V_{POUT} = V_{POUT,inf} + dV_{POUT}, \tag{5.6}$$

where $V_{POUT,inf}$ is the output voltage for infinite CMRR (i.e., without the CMRR effect).

These effects incur the variance at around the threshold (i.e., zero total sum). In BNN, since the output activation is binary, the variance at around the threshold is very important. The importance is doubled since the occurrence of zero sum is the most frequent [7]. On the other hand, if the total sum is much larger or smaller than the threshold, the output can be correctly determined regardless of the variance. Assuming that Figure 5.7a is for the total sum, the variance in V_{POUT} around zero sum is rather wide and overlapped. Thus, V_{POUT} for a positive sum (say 1) can be less than the threshold (say 1.095V in our example) in some case, resulting in the output activation

of 0 rather than 1. The Figure 5.7a shows that the minimal V_{POUT} variation is at around -3 not 0. It is because the directions of voltage variations for $V_{POUT,inf}$ and dV_{POUT} are opposite in the negative partial sum domain. Moreover, when the process variation is considered, the minimal point shifts as shown in Figure 5.8a. These effects make PS32 vulnerable to incorrect decisions near the threshold resulting in severe test accuracy drop.

5.3 Variation Compensation

Analog systems have advantages of fast operation speed, small energy consumption, and compact area compared to digital systems. However, a critical weakness of the analog systems is vulnerability to noise [82]. One of the biggest noise sources is a process variation. It is a big obstacle to using analog circuits since the variation shifts various parameters of analog circuits from correct operating points which causes severe performance degradation or even malfunction. In this section, we describe about the model of process variation used in this work and a solution to restore the classification accuracy hurt by process variation.

5.3.1 Variation Modeling

Process variations can be categorized as local and global components. The local variation affects every device in a chip differently. It mainly comes from the random fluctuation of dopant distributions in device channels [83] and the line edge roughness of gates [84]. On the other hand, the global variation affects differently across chips. The different process corners (SS, TT, FF, etc.) are due to the global variation, and its main contributors are the fluctuations of gate width and length, oxide thickness, and channel dopant concentration [85]. Those parameter variations result in the variation of threshold voltage V_T, which affects transistor performance significantly [85, 86].

We model the process variation similar to the method used in [85, 87] by using the threshold voltage variation since an accurate modeling of the process variation is not the scope of this study. We use a normal distribution to model the global variation of V_T which affects differently across chips. To model the random impact caused by the local variation, we use another normal distribution of V_T which affects every transistor differently in a chip. These variations can be formulated as

$$dV_{T,global} = \mathcal{N}(0,\ \sigma^2_{V_{T,global}}), \tag{5.7}$$

$$dV_{T,local} = \mathcal{N}(0,\ \sigma^2_{V_{T,local}}), \tag{5.8}$$

$$V_T = V_{T0} + dV_{T,global} + dV_{T,local}, \tag{5.9}$$

where $\sigma_{V_{T,global}}$ and $\sigma_{V_{T,local}}$ are standard deviations of the global and local variations, and V_{T0} is the default threshold voltage of typical corner characterized in a technology library.

We also account for the resistance variations that can be caused by process variation and ReRAM programming inaccuracy [29]. The resistance R affected by process variation is modeled as

$$dR_{global} = \mathcal{N}(0,\ \sigma^2_{R_{global}}), \tag{5.10}$$

$$dR_{local} = \mathcal{N}(0,\ \sigma^2_{R_{local}}), \tag{5.11}$$

$$R = R_0 + dR_{global} + dR_{local}, \tag{5.12}$$

where R_0 is the target resistance, and $\sigma_{R_{global}}$ and $\sigma_{R_{local}}$ are standard deviations of the corresponding resistance variations.

The ReRAM resistance R_p affected by programming inaccuracy approximately follows log-normal distribution [29], thus we model it as

$$ln(R_p) = \mathcal{N}(\mu_p,\ \sigma_p^2). \tag{5.13}$$

In LRS programming, the target resistance is $2\,\mathrm{M}\Omega$ and its corresponding μ_p is set accordingly. The standard deviation σ_p is set differently depending on the variation scenario as shown in Table 5.3. HRS programming suffers from more significant variation compared to LRS programming. However, the R_{HRS} variation can be ignored if the target resistance is set large enough (over $50\,\mathrm{M}\Omega$) such that the current through R_{HRS} is much smaller than that through R_{LRS}.

5.3.2 Impact of V_T Variation

The V_T variation changes operating characteristics of analog circuits. Figure 5.8a shows the voltage outputs of PS32 under different V_T variations, which is simulated by using a circuit simulator with 32nm technology library. Depending on the V_T variation, the output voltage curve shifts and its slope also changes, which affects output activations of neurons.

Figure 5.8b and 5.8c plot average activation values of randomly selected neurons of CNN on CIFAR-10 described in Table 5.1. The average activation value $\hat{a}_i$ of i-th neuron is calculated by taking an average over all output activation values of the neuron when inferences are performed with all images in a training set. It can be formulated as

$$\hat{a}_i = \frac{1}{N} \sum_{k}^{I} a_i(k) \tag{5.14}$$

where N is the number of images in a training set I and $a_i(k)$ is an output activation value of i-th neuron when the image k is applied as an input. If the process variation is applied toward the direction of increasing V_T, output voltage of PS32 decreases and thus average activation values of the neuron also decrease in a bottom layer (red line in Figure 5.8b). On the other hand, if the process variation leads to a decrease of V_T, the output voltage curve of PS32 goes up and it increases the average activation values (green line in Figure 5.8b). The distorted activations in the bottom layer are propagated into top layers, and they make the average activation values of neurons stuck at 0 or 1 (Figure 5.8c). These erroneous operations occur in many neurons in a network, which severely hurts the classification accuracy of the network as shown in Figure 5.9 and 5.10.

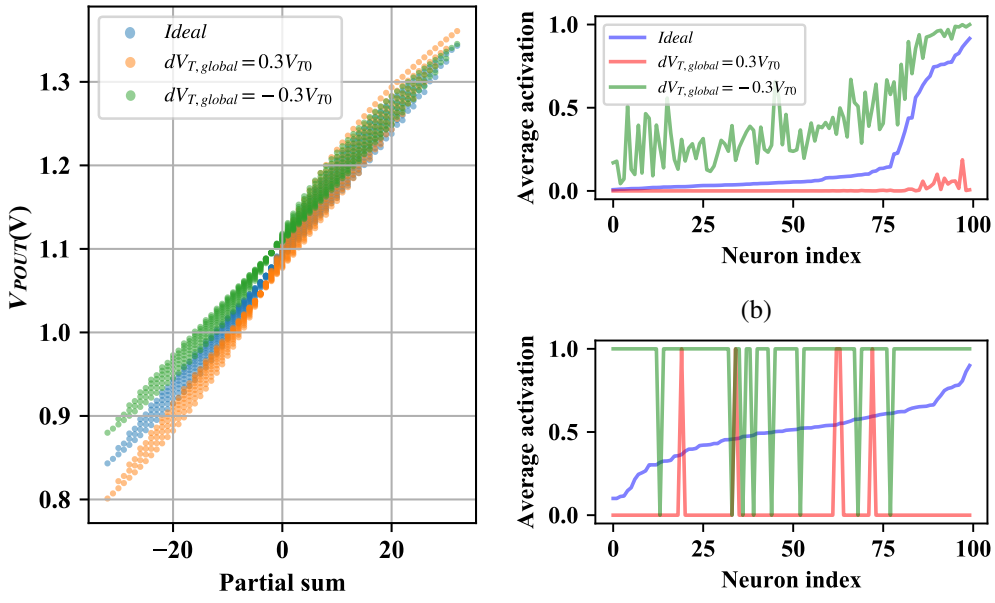


Figure 5.8: (a) Output voltages of PS32 under V_T variation. (b) Average activation values of 100 random neurons under V_T variation in a bottom layer and (c) in a top layer.

5.3.3 Variation Compensation Techniques

Due to the severe degradation caused by variations, a variation compensation technique is inevitably required in order to use the proposed analog circuits.

Global shift of V_{ref} and Bias

As shown in Figure 5.8a, the difference in V_{POUT} for the same partial sum, which is originated from the nonlinearity and finite CMRR of the differential amplifier in PS32, is minimized when the partial sum value is between -1 and -4 (instead of 0) depending on the V_T variation. Since the occurrence of zero partial sum is the most frequent during inference, we adjust both V_{ref} and the bias values of PS32 accordingly so that PS32 can operate most frequently at the point where the fluctuation of V_{out} is small. If

the bias value of the PS circuit is lowered and V_{ref} is adjusted lower accordingly, PS32 can operate at a point where the non-linearity due to the CMRR issue is smallest. The shift of V_{ref} and bias is performed identically for each neuron and PS32. According to our experiments, this simple technique improves classification accuracy significantly. However, it is not enough for variation compensation since the local variation, which differentiates V_{POUT} curve of PS32 for each neuron, is not correctly handled by the technique.

Activation Matching via Bias Tuning

The shift of V_{POUT} curve of PS32 due to V_T variation changes average activation values of neurons, which is a direct cause of degraded classification accuracy. Although the V_T variation distorts the average activation value, the monotonicity of the circuit is still maintained. Thus, if a V_T variation causes a drop of average activation value of a neuron, it can be compensated by decreasing V_{ref} of the neuron. We observe that the classification accuracy of a network is restored if the distortions of average activation values of neurons are compensated to have the same average activation values as the ideal case in which no V_T variation is applied. Therefore, if V_{ref} of each neuron is adjusted properly, the proposed analog hardware performs well even in the presence of severe process variation. However, this method is somewhat impractical since it is difficult to implement a system that adjusts V_{ref} of each neuron differently.

One way to compensate the V_T variation is adjusting the V_{ref} properly for each neuron. Although the V_T variation distorts the output voltage of a neuron, the increasing trend of the output voltage of a neuron is still maintained when the weighted input sum increases. When the V_T variation is not applied, a neuron produces a certain number of high output voltages if inferences are performed with all the images in a training set. Thus, if we adjust the V_{ref} of each neuron to a level that results in the same number of high output voltages with the inferences for the same training set, the degraded classification accuracy due to the V_T variation can be restored. However, this method

is somewhat impractical since it is difficult to implement a system that adjusts the V_{ref} of every neuron precisely.

Another solution for variation compensation, that is much more practical, is adjusting the ReRAM cells representing a bias instead of the direct V_{ref} adjustment. In a neuron circuit integrating multiple PS32s, most ReRAM cells are used to represent synaptic weights of the neuron and the remaining cells are used to represent a multi-bit bias of the neuron. Since each neuron has various bias values, the ReRAM cells for a bias must be large enough to represent the largest bias value. Thus, a neuron with the non-largest bias value may has unused ReRAM cells, and the unused cells can be exploited for variation compensation. In this technique, some synapses of PS32 are reserved for the variation compensation and the other synapses are used to represent the weights and biases. The effect of decreasing/increasing V_{ref} can be obtained by programming some of the unused ReRAM cells to have value +1/-1. On the other hand, the effect of increasing V_{ref} can be realized by programming some unused ReRAM cells toward "-1".

Algorithm 2 demonstrates the procedure of the variation compensation technique through activation matching (VCAM). Basically, it adjusts all biases in a layer simultaneously with Binary Search. At first, the ReRAM synaptic arrays of a sample chip C are programmed with learned weights $\boldsymbol{W}_0$ and biases $\boldsymbol{B}_0$ (line 1), and current biases $\boldsymbol{B}$ are initialized with $\boldsymbol{B}_0$ (line 2). For each layer l in a network (line 1), the maximum possible bias value of the layer is stored to M^l (line 2) and the initial step size is set to $2^{\lceil log_2 M \rceil - 1}$ in order to make the biases reachable to any value in a possible bias range (line 3). Inferences are performed with input images $\boldsymbol{I}$ on the chip C, and then the average activation values $\boldsymbol{A}^l$ of all neurons in the layer l are computed (line 5). The differences ($\Delta \boldsymbol{A}^l$) of average activation values between the physical circuit ($\boldsymbol{A}^l$) and the trained model ($\boldsymbol{A}^l_T$) are calculated (line 6) and the biases in layer l ($\boldsymbol{B}^l$) are adjusted by $-Sign(\Delta \boldsymbol{A}^l) \cdot step$ (line 7). Then the ReRAM arrays corresponding to the updated biases are reprogrammed (line 8), and the step size is reduced by half (line

Algorithm 2 Variation compensation through activation matching (VCAM):

Input: a sample chip C, input images $\boldsymbol{I}$ in a training set, learned weights $\boldsymbol{W}_0$ and biases $\boldsymbol{B}_0$, # of layers L in a network, and average activation values of neurons for L layers $\boldsymbol{A}_T^1, \boldsymbol{A}_T^2, \ldots, \boldsymbol{A}_T^L$ over input images $\boldsymbol{I}$ in the trained model.

Output: a sample chip C with restored accuracy after bias tuning.

1: Program_ReRAM($C, \boldsymbol{W}_0, \boldsymbol{B}_0$)
2: $\boldsymbol{B} \leftarrow \boldsymbol{B}_0$
3: **for** $l \leftarrow 1$ **to** L **do**
4: $M^l \leftarrow$ maximum possible bias value
5: $step \leftarrow 2^{\lceil log_2 M^l \rceil - 1}$
6: **while** $step > 0$ **do**
7: $\boldsymbol{A}^l \leftarrow$ Compute_average_activation($C, l, \boldsymbol{I}$)
8: $\Delta \boldsymbol{A}^l \leftarrow \boldsymbol{A}^l - \boldsymbol{A}_T^l$
9: $\boldsymbol{B}^l \leftarrow \boldsymbol{B}^l - Sign(\Delta \boldsymbol{A}^l) \cdot step$
10: Reprogram_ReRAM($C, \boldsymbol{B}^l$)
11: $step \leftarrow step/2$

9). The stages 5-9 are repeated until the step size becomes zero (line 4).

After the activation matching process, every neuron in the sample chip will have average activation value similar to that in the ideal case, which restores the classification accuracy corrupted by the process variation.

Overall compensation process

The variation compensation technique incurs an area overhead due to the redundant synaptic arrays for the bias control. According to our experiments, about 10% of more synapses are enough for the variation compensation. These techniques also require an off-chip system that feeds image inputs into the sample chip, gathers activations of neurons from the chip, and programs ReRAM synapses in the chip. If a chip is

fabricated, various V_{ref} is tested with a few bias shift candidates (e.g., -5 $\sim$ 0), and then the V_{ref} and bias shift value leading to the best accuracy are selected. After that, the activation matching process is performed to adjust the biases of each neuron differently for further accuracy improvement. Although the process requires many inferences with training set images, it does not take long time since an inference with a fabricated physical chip can be performed quickly.

5.4 Experimental Results

5.4.1 Experimental Setup

Table 5.1: Network topologies for MNIST, CIFAR-10, and ImageNet

Layer #	MLP on MNIST	CNN on CIFAR-10	CNN on ImageNet
1	FC (784, 512)	3x3 Conv. (3, 128)	11x11 Conv. (3, 64)
2	FC (512, 512)	3x3 Conv. (128, 128)	3x3 Max pooling
3	FC (512, 512)	2x2 Max pooling	5x5 Conv. (64, 192)
4	FC (512, 10)	3x3 Conv. (128, 256)	3x3 Max pooling
5	-	3x3 Conv. (256, 256)	3x3 Conv. (192, 384)
6	-	2x2 Max pooling	3x3 Conv. (384, 256)
7	-	3x3 Conv. (256, 512)	3x3 Conv. (256, 256)
8	-	3x3 Conv. (512, 512)	FC (9216, 4096)
9	-	2x2 Max pooling	FC (4096, 4096)
10	-	FC (8192, 1024)	FC (4096, 1000)
11	-	FC (1024, 1024)	-
12	-	FC (1024, 10)	-

Experiments were performed with MLP and CNN on MNIST, CIFAR-10, and ImageNet datasets. The network topologies are described in Table 5.1. A multi-layer perceptron with three hidden layers (each layer has 512 neurons) is used for experiments on MNIST. In the CNN on CIFAR-10, a convolutional neural network including 6 convolution layers, 3 max pooling layers, and 3 fully-connected layers is used for experiments on CIFAR-10. Every convolution layer uses 3x3 filters with stride 1 and one zero padding. In the CNN on ImageNet, the first convolution layer uses 11x11 filters with stride 4, the second one uses 5x5 filters with stride 1, and the others use 3x3 filters with stride 1. The two numbers in the parenthesis for each fully-connected or convolu-

tion layer mean the number of input and output neurons. Since the proposed hardware architecture cannot handle multi-bit inputs, the first layer of a network is not covered.

We used the programming framework PyTorch and NVIDIA TITAN-X GPU to train the networks. Although the batch normalization layers were used for the successful network training, they were removed in the hardware implementation by merging the BN parameters on the biases as explained in Section 5.1.3. Since a whole network is too large to evaluate with a circuit simulator, we characterized the proposed neuron circuit by using HSPICE circuit simulator with 32nm Synopsys generic library, and then evaluated the proposed technique based on the characterized information in a PyTorch program. To simulate process variation, Monte Carlo simulation was used to fluctuate the threshold voltages of all transistors and the resistances of all ReRAM and resistors to normal distributions.

5.4.2 Accuracy of the Modified BNN Algorithm

Table 5.2: Accuracy comparison among BNN algorithms

	BNN	*BNN-mod1*	*BNN-mod2*	*BNN-mod3*
MLP (MNIST)				
MIN	98.42%	98.43%	98.54%	98.54%
MAX	98.63%	98.60%	98.66%	98.63%
AVG.	**98.51%**	**98.51%**	**98.62%**	**98.60%**
CNN (CIFAR10)				
MIN	89.08%	89.96%	90.82%	90.80%
MAX	89.80%	90.61%	91.24%	91.13%
AVG.	**89.51%**	**90.33%**	**91.04%**	**90.98%**
CNN (ImageNet)	**64.68%**	**64.67%**	**65.19%**	**64.72%**

- Numbers for ImageNet show top-5 accuracy.

Table 5.2 compares classification accuracies of networks trained with different

BNN algorithms on MNIST and CIFAR-10. For each algorithm on MNIST and CIFAR-10, trainings are performed 10 times with random seeds , and the minimum, maximum, and average of the classification accuracies are calculated. The training for each algorithm on ImageNet is performed once due to the long runtime overhead. The algorithm *BNN* indicates the original BNN algorithm [2]. *BNN-mod1* is the modified BNN algorithm which uses 0 and 1 activations instead of -1 and 1. *BNN-mod2* uses 0 and 1 activations and constrains the minimum of γ to 0.5, and *BNN-mod3* additionally fixes the scale factors of the last batch normalization layer from *BNN-mod2*. Each modification of BNN algorithm shows similar classification accuracy to the original BNN algorithm. Thus, we use *BNN-mod3* algorithm for hardware implementation since it is the most advantageous way to reduce hardware cost without accuracy loss.

5.4.3 Variation Compensation

Table 5.3: Variation scenarios

	Tiny var.	Small var.	Medium var.	Large var.
$3\sigma_{V_{T,global}}$	5% of V_{T0}	10% of V_{T0}	30% of V_{T0}	50% of V_{T0}
$3\sigma_{V_{T,local}}$	0	2.5% of V_{T0}	7.5% of V_{T0}	12.5% of V_{T0}
$3\sigma_{R_{global}}$	2.5% of R_0	5% of R_0	10% of R_0	15% of R_0
$3\sigma_{R_{local}}$	0	1% of R_0	2% of R_0	3% of R_0
σ_p	0.008	0.016	0.033	0.050

To evaluate performance of the proposed circuit under variations, we modeled four different process variation scenarios as shown in Table 5.3. The tiny, small, medium, and large variation scenarios have 5%, 10%, 30%, and 50% global threshold voltage variations at 3σ point, respectively. Since the amount of local variation is usually much lower than that of global variation, the local threshold variation is set to $1/4$ of the global variation for every scenario based on the measurement results in [88]. The

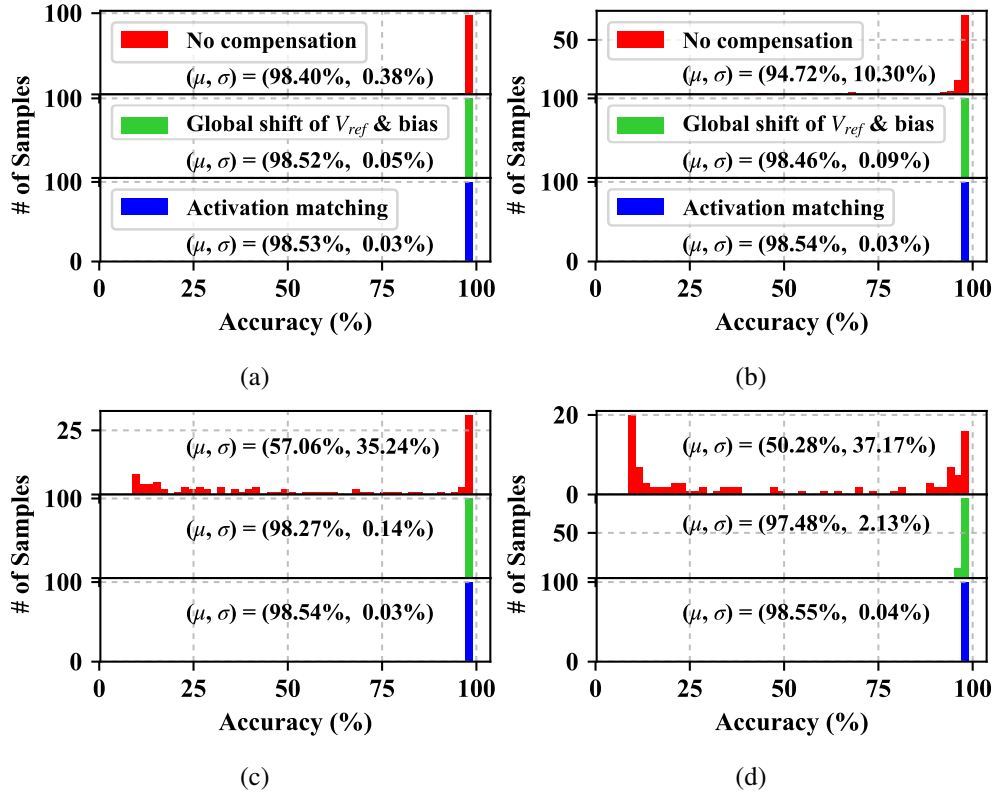


Figure 5.9: Accuracy distributions of MLP on MNIST under (a) tiny, (b) small, (c) medium, and (d) large variation scenario.

resistors are also affected by process variation. In case of ReRAM, there is another resistance variation caused by inaccurate ReRAM programming which is independent from the process variation. The amount of resistance variations are based on the results in [29, 89]. The σ_p values are set such that the standard deviation of target programming resistance (R_{p0}) at 3-sigma is 2.5%, 5%, 10%, and 15% of R_{p0} under tiny, small, medium, and large variation scenarios, respectively. For each variation scenario, we performed 100 Monte Carlo simulations to emulate 100 sample chips that are affected by different variations.

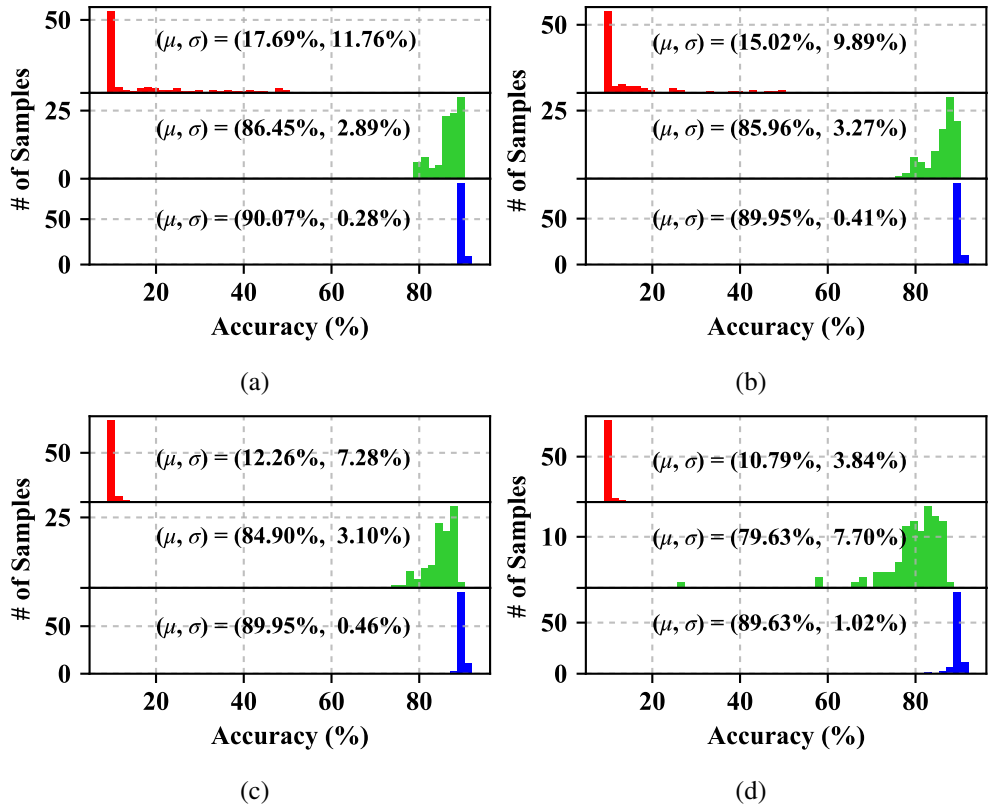


Figure 5.10: Accuracy distributions of CNN on CIFAR-10 under (a) tiny, (b) small, (c) medium, and (d) large variation scenario.

Figure 5.9 and 5.10 compare classification accuracy distributions of MLP and CNN when the proposed variation compensation technique is applied under the four different process variation scenarios. Before the variation compensation techniques are applied, the classification accuracies of MLP and CNN are severely degraded by the variations. Since CIFAR-10 dataset is much more complex than MNIST, CNN on CIFAR-10 is more vulnerable to the variations. The classification accuracy of CNN is severely degraded from 91.10% (accuracy when the variations are not considered) to 15.02% on average even under the small variation scenario, which makes the neural network infeasible. However, the accuracy distributions are dramatically improved (1.35% drop from 90.98%) after the proposed variation compensation techniques are

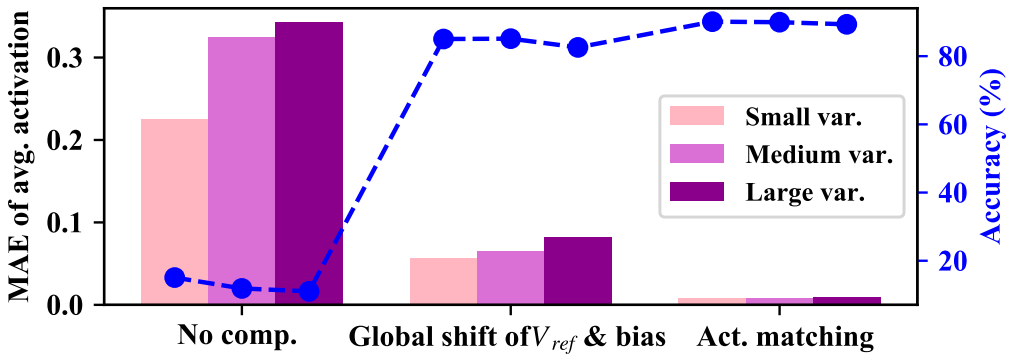


Figure 5.11: Mean absolute error (MAE) of average activation values and classification accuracies of CNN on CIFAR-10 under different variation scenarios.

performed. In MLP on MNIST, the average accuracy is restored to 98.55% which is close to the accuracy 98.63% in the ideal case. In CNN on CIFAR-10, the average accuracy becomes 89.63% under large variation scenario, which is just 1.5% drop from the theoretical accuracy 91.13%.

Under the tiny variation scenario of CNN on CIFAR-10, the global tuning of V_{ref} and bias does not yield a good result (86.45% accuracy on average) even if the local variation is not applied. It is because the precision of V_{ref} control is set to 1mV. If V_{ref} can be controlled more precisely, every ample chip can operate with a classification accuracy of 89% or more, and the average classification accuracy goes up over 89.4%.

In case of the tiny variation scenario in which the local variation is zero or negligibly small, the variation compensation can be done by using a monitoring circuit which measures the amount of global variation as the work [90]. We can implement the scheme by inserting a monitoring PS32 circuit neuron of which output voltage is able to be measured. If the monitoring PS32 is configured to operate with zero partial sum condition, its output voltage, which is affected by the global variation, can be a guide for V_{ref} of comparators. This approach based on a monitoring circuit simplifies the compensation scheme to avoid the large calibration time required by the activation matching process.

Figure 5.11 shows the difference between the average activation values of all neurons and those of the ideal neurons in mean absolute error (MAE) and classification accuracies of CNN on CIFAR-10 under different variation scenarios. The larger variation causes the larger error of average activations, and the classification accuracy is improved if the error is reduced by the global shift of V_{ref} and bias (see Section 5.3.3), and the proposed activation matching techniques.

5.4.4 Performance Comparison

Table 5.4: Comparison among BNN implementations

Work	Net	# Param	Tech	Latency	Area (mm^2)	Avg. Power (mW)	Energy (nJ)	Energy Efficiency (TOPS/W)
[32]	CNN	1.26M	45nm	-	0.06	-	13550	0.962*
[31]	CNN	0.26M	65nm	-	3.61	-	79.72	0.048
[29]	CNN	14.03M	40nm	1.6ms	1.02	6.3	9815*	126
[6]	CNN	1.88M	28nm	4.2ms	5.76	0.9	3790	532
[7]	MVM	0.07M	65nm	13.69ns	0.05	67.9*	0.93*	141
Ours	MVM	0.07M	32nm	0.7ns	0.02	169.1	0.12	1107
Ours	MLP	0.53M	32nm	2.1ns	0.15	519.6	1.09	970

The data marked with '*' is calculated based on the numbers in the paper.

Table 5.4 represents the performance comparison result among different BNN implementations. We simulated 256x256 matrix-vector-multiplication (MVM) and the MLP on MNIST by using Synopsys FineSim circuit simulator.[3] Since we fully parallelize the network, the whole synapses and neurons in a layer operate simultaneously and consume large power compared to the previous works. However, due to the parallel operation of the differential amplifiers producing partial sum outputs within 0.4ns,

[3]The entire CNN on CIFAR-10 was too large to be evaluated with the circuit simulator.

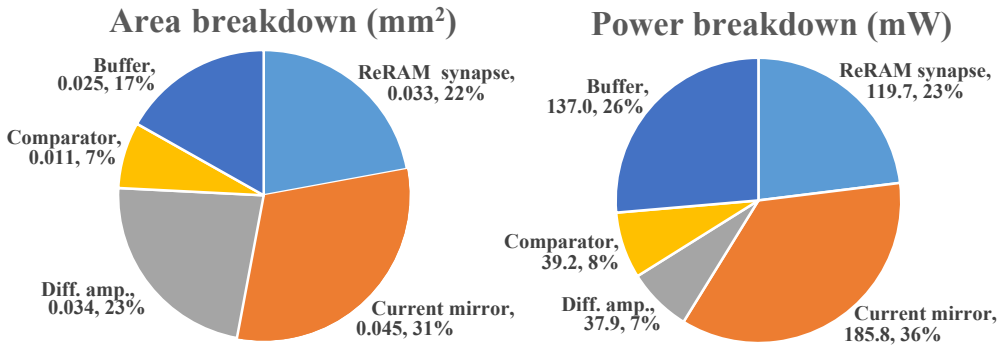


Figure 5.12: Area and power breakdown results of MLP.

the computation of 256x256 MVM can be done within 0.7ns, which is much faster than [7] exploiting digital logic to add partial sums. Thanks to the short latency, MLP with the proposed architecture achieves 970 TOPS/W energy efficiency, which is much better than those reported previously. Although the different network topologies in Table 5.4 make fair comparison difficult, energy efficiency can be considered as a good metric for relatively fair comparison.

Area and power breakdown results of MLP are shown in Figure 5.12. It consists of 1235 ReRAM tiles (32x32) and 1034 neuron circuits. The neuron circuits consume 2-3 times more area and power than ReRAM synapses. Many standard cell buffers used for numerous neuron-to-neuron interconnections also take considerable amount of area and power consumption.

5.5 Summary

We proposed a BNN hardware architecture with ReRAM synapses and analog neurons based on differential amplifiers, and a novel variation compensation technique through activation matching (VCAM). The experimental results show that the proposed variation compensation technique effectively restores the classification accuracy degraded by process variation, and thus makes the analog BNN implementation feasible. Due to the use of analog synapses and neurons, the proposed architecture achieves very high energy efficiency. Although this work accounts for the process and ReRAM resistance variations, it can also be extended to compensate temperature variation by adjusting the biases dynamically according to the change of temperature.

Chapter 6

Conclusion

Deep learning has achieved great successes in image classification and many other areas. However, they accompany tremendous number of computations and excessive power consumption, which limits the broad use of deep learning especially in embedded systems. Thus, lowering the energy consumption of deep learning is imperative. To cope with the problem, techniques for designing energy efficient neural networks are proposed

First, we proposed a novel spiking neural network with weighted spikes to overcome the slow classification problem of the conventional SNNs with rate coding. The key idea is assigning different weights to spikes depending on the time phase within a period to encode more information within less number of spikes. The proposed SNN-WS reduces the classification latency as well as number of spikes significantly. The approach can be applied to various types of networks including deep residual networks and various datasets without accuracy loss. We also proposed an early decision algorithm to further reduce latency and number of spikes at the expense of small accuracy loss.

Second, we proposed a BNN hardware architecture with ReRAM synapses and analog neurons based on differential amplifiers, and a novel variation compensation technique through activation matching (VCAM). The experimental results show that the proposed variation compensation technique effectively restores the classification accuracy degraded by process variation, and thus makes the analog BNN implementation feasible. Due to the use of analog synapses and neurons, the proposed architecture achieves very high energy efficiency. Although this work accounts for the process and ReRAM resistance variations, it can also be extended to compensate temperature variation by adjusting the biases dynamically according to the change of temperature.

Bibliography

[1] K. Hwang and W. Sung, "Fixed-point feed forward deep neural network design using weights +1, 0, and -1," in *IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.

[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv*, 2016.

[3] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2237–2251, 2010.

[4] S. Yu, B. Lee, and H.-S. P. Wong, "Metal oxide resistive switching memory," in *Functional Metal Oxide Nanostructures*. Springer, 2012, pp. 303–335.

[5] S. Park, S. Kim, H. Choe, and S. Yoon, "Fast and efficient information transmission with burst spikes in deep spiking neural networks," *arXiv preprint arXiv:1809.03142*, 2018.

[6] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS," in *Proceedings of International Solid-State Circuits Conference (ISSCC)*, 2018, pp. 222–224.

[7] X. Sun, S. Yin, X. Peng, R. Liu, J.-s. Seo, and S. Yu, "XNOR-RRAM: A scal-

able and parallel resistive synaptic architecture for binary neural networks," in *Proceedings of Design, Automation and Test in Europe*, 2018, pp. 1423–1428.

[8] J. Kim, C. Lee, and K. Choi, "Energy efficient analog synapse/neuron circuit for binarized neural networks," in *International SoC Design Conference*, Nov. 2018.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, Dec. 2012, pp. 1097–1105.

[10] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Oct. 2012.

[11] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.

[12] "GPU-based deep learning inference: A performance and power analysis," Nvidia White paper, 2015.

[13] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *Proceedings of Custom Integrated Circuits Conference (CICC)*, 2011, pp. 1–4.

[14] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, "Energy-efficient neuron, synapse and STDP integrated circuits," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 3, pp. 246–256, 2012.

[15] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million

spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[16] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[17] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[18] J. Vreeken *et al.*, "Spiking neural networks, an introduction," Institute for Information and Computing Sciences, Utrecht University Technical Report, 2002.

[19] D. Heeger, "Poisson model of spike generation," http://www.cns.nyu.edu/ david/handouts/poisson.pdf, 2000.

[20] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and tools for the conversion of analog to spiking convolutional neural networks," in *NIPS Workshop on Computing with Spikes*, 2016.

[21] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, "Deep neural networks with weighted spikes," *Neurocomputing*, vol. 311, pp. 373–386, 2018.

[22] J. Lisman and G. Buzsáki, "A neural coding scheme formed by the combined function of gamma and theta oscillations," *Schizophrenia Bulletin*, vol. 34, no. 5, pp. 974–980, 2008.

[23] C. Kayser, M. A. Montemurro, N. K. Logothetis, and S. Panzeri, "Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns," *Neuron*, vol. 61, no. 4, pp. 597–608, 2009.

[24] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading

digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[25] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech Report, 2009.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[27] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," in *Proceedings of Advances in Neural Information Processing Systems*, 2017.

[28] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC," in *International Conference on Field-Programmable Technology (FPT)*, 2016, pp. 77–84.

[29] L. Ni, Z. Liu, W. Song, J. J. Yang, H. Yu, K. Wang, and Y. Wang, "An energy-efficient and high-throughput bitwise CNN on sneak-path-free digital ReRAM crossbar," in *International Symposium on Low Power Electronics and Design*, 2017, pp. 1–6.

[30] X. Sun, X. Peng, P.-Y. Chen, R. Liu, J.-s. Seo, and S. Yu, "Fully parallel RRAM synaptic array for implementing binary neural network with (+1, -1) weights and (+1, 0) neurons," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, 2018, pp. 574–579.

[31] D. Miyashita, S. Kousai, T. Suzuki, and J. Deguchi, "A neuromorphic chip optimized for deep learning and CMOS technology with time-domain analog and digital mixed-signal processing," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 10, pp. 2679–2689, 2017.

[32] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on RRAM," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 782–787.

[33] S. Mittal, "A survey of ReRAM-based architectures for processing-in-memory and neural networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, 2018.

[34] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609–622.

[35] J. S. Meena, S. M. Sze, U. Chand, and T.-Y. Tseng, "Overview of emerging nonvolatile memory technologies," *Nanoscale Research Letters*, vol. 9, no. 1, p. 526, 2014.

[36] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[37] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[38] M. Fujimoto, H. Koyama, M. Konagai, Y. Hosoi, K. Ishihara, S. Ohnishi, and N. Awaya, "TiO2 anatase nanolayer on TiN thin film exhibiting high-speed bipolar resistive switching," *Applied physics letters*, vol. 89, no. 22, p. 223509, 2006.

[39] W. M. Kistler, "Spike-timing dependent synaptic plasticity: a phenomenological framework," *Biological Cybernetics*, vol. 87, no. 5, pp. 416–427, 2002.

[40] Y. Dan and M.-m. Poo, "Spike timing-dependent plasticity of neural circuits," *Neuron*, vol. 44, no. 1, pp. 23–30, 2004.

[41] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Computational Biology*, vol. 3, no. 2, p. e31, 2007.

[42] T. J. Strain, L. McDaid, T. M. McGinnity, L. P. Maguire, and H. M. Sayers, "An STDP training algorithm for a spiking neural network with dynamic threshold neurons," *International Journal of Neural Systems*, vol. 20, no. 06, pp. 463–480, 2010.

[43] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers in neuroscience*, vol. 7, p. 272, 2014.

[44] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015.

[45] T. Iakymchuk, A. Rosado-Muñoz, J. F. Guerrero-Martínez, M. Bataller-Mompeán, and J. V. Francés-Víllora, "Simplified spiking neural network architecture and STDP learning algorithm applied to image classification," *EURASIP Journal on Image and Video Processing*, vol. 2015, no. 1, pp. 1–11, 2015.

[46] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 9–48.

[47] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[48] B. Schrauwen and J. Van Campenhout, "Extending spikeprop," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, vol. 1, 2004, pp. 471–475.

[49] S. M. Silva and A. E. Ruano, "Application of levenberg-marquardt method to the training of spiking neural networks," in *International Conference on Neural Networks and Brain*, vol. 3, 2005, pp. 1354–1358.

[50] S. McKennoch, D. Liu, and L. G. Bushnell, "Fast modifications of the spikeprop algorithm," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2006, pp. 3970–3977.

[51] S. Ghosh-Dastidar and H. Adeli, "Improved spiking neural networks for eeg classification and epilepsy and seizure detection," *Integrated Computer-Aided Engineering*, vol. 14, no. 3, pp. 187–212, 2007.

[52] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[53] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *IRE WESCON Convention Record*, 1960, pp. 96–104.

[54] B. Widrow, "Generalization and information storage in network of adaline neurons," *Self-organizing Systems*, pp. 435–462, 1962.

[55] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "DL-ReSuMe: A delay learning-based remote supervised method for spiking neurons," *Transactions on Neural Networks and Learning Systems*, vol. 26, no. 12, pp. 3137–3149, 2015.

[56] Y. Xu, X. Zeng, and S. Zhong, "A new supervised learning algorithm for spiking neurons," *Neural Computation*, vol. 25, no. 6, pp. 1472–1511, 2013.

[57] X. Xie, H. Qu, G. Liu, and M. Zhang, "Efficient training of supervised spiking neural networks via the normalized perceptron based learning rule," *Neurocomputing*, vol. 241, pp. 152–163, 2017.

[58] Z. Lin, D. Ma, J. Meng, and L. Chen, "Relative ordering learning in spiking neural network for pattern recognition," *Neurocomputing*, 2017.

[59] Y. LeCun, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[60] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in Neuroscience*, vol. 7, 2013.

[61] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2706–2719, 2013.

[62] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, May 2015.

[63] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 15, 2011, pp. 315–323.

[64] E. D. Adrian, "The impulses produced by sensory nerve endings," *The Journal of physiology*, vol. 61, no. 1, pp. 49–72, 1926.

[65] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural networks*, vol. 14, no. 6, pp. 715–725, 2001.

[66] A. Belatreche, L. Maguire, M. McGinnity, and Q. Wu, "A method for supervised training of spiking neural networks," in *Proceedings of IEEE Conference on Cybernetics Intelligence–Challenges and Advances (CICA)*, 2003, pp. 39–44.

[67] S. J. Thorpe, "Spike arrival times: A highly efficient coding scheme for neural networks," *Parallel Processing in Neural Systems*, pp. 91–94, 1990.

[68] E. M. Izhikevich, "Resonance and selective communication via bursts in neurons having subthreshold oscillations," *BioSystems*, vol. 67, no. 1, pp. 95–102, 2002.

[69] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, 2015, pp. 448–456.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[71] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proceedings of the International Conference on Learning Representations*, 2014.

[72] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[73] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*. John Wiley & Sons, 2004.

[74] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, 2015, pp. 562–570.

[75] F. Chollet, “Keras,” https://github.com/fchollet/keras, 2015.

[76] A. Vedaldi and K. Lenc, “MatConvNet: Convolutional neural networks for MATLAB,” in *Proceeding of the ACM International Conference on Multimedia*, 2015.

[77] E. Anderson, “The species problem in iris,” *Annals of the Missouri Botanical Garden*, vol. 23, no. 3, pp. 457–509, 1936.

[78] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, 2015, pp. 448–456.

[79] M.-F. Chang, S.-S. Sheu, K.-F. Lin, C.-W. Wu, C.-C. Kuo, P.-F. Chiu, Y.-S. Yang, Y.-S. Chen, H.-Y. Lee, C.-H. Lien *et al.*, “A high-speed 7.2-ns read-write random access 4-mb embedded resistive RAM (ReRAM) macro using process-variation-tolerant current-mode read schemes,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 878–891, 2013.

[80] M. Mao, Y. Cao, S. Yu, and C. Chakrabarti, “Optimizing latency, energy, and reliability of 1T1R ReRAM through cross-layer techniques,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 3, pp. 352–363, 2016.

[81] P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser, and D. Modha, “Deep neural networks are robust to weight binarization and other non-linear distortions,” *arXiv preprint arXiv:1606.01981*, 2016.

[82] R. Sarpeshkar, “Analog versus digital: extrapolating from electronics to neurobiology,” *Neural Computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[83] P. A. Stolk *et al.*, “Modeling statistical dopant fluctuations in MOS transistors,” *IEEE TED*, vol. 45, no. 9, pp. 1960–1971, 1998.

[84] J. Croon *et al.*, "Line edge roughness: characterization, modeling and impact on device behavior," in *Proc. IEDM*, 2002.

[85] S. Garg and D. Marculescu, "Addressing process variations at the microarchitecture and system level," *Foundations and Trends in Electronic Design Automation*, vol. 6, pp. 217–291, 2013.

[86] M. J. Pelgrom *et al.*, "Matching properties of MOS transistors," *IEEE JSSC*, vol. 24, no. 5, pp. 1433–1439, 1989.

[87] K. A. Bowman, S. G. Duvall, and J. D. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, pp. 183–190, 2002.

[88] L.-T. Pang, K. Qian, C. J. Spanos, and B. Nikolic, "Measurement and analysis of variability in 45 nm strained-Si CMOS technology," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 8, pp. 2233–2243, 2009.

[89] T. Lin, Y. Ho, and C. Su, "High-R poly resistance deviation improvement from suppressions of back-end mechanical stresses," *IEEE Transactions on Electron Devices*, vol. 64, no. 10, pp. 4233–4241, 2017.

[90] J. Kim, K. Choi, Y. Kim, W. Kim, K. Do, and J. Choi, "Delay monitoring system with multiple generic monitors for wide voltage range operation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 37–49, 2018.

초 록

최근 심층 학습은 이미지 분류, 음성 인식 및 강화 학습과 같은 영역에서 놀라운 성과를 거두고 있다. 최첨단 심층 인공신경망 중 일부는 이미 인간의 능력을 넘어선 성능을 보여주고 있다. 그러나 인공신경망은 엄청난 수의 고정밀 계산과 수백만개의 매개 변수를 이용하기 위한 빈번한 메모리 액세스를 수반한다. 이는 엄청난 칩 공간과 에너지 소모 문제를 야기하여 임베디드 시스템에서 인공신경망이 사용되는 것을 제한하게 된다. 이 문제를 해결하기 위해 인공신경망을 높은 에너지 효율성을 갖도록 설계하는 방법을 제안한다.

첫번째 파트에서는 가중 스파이크를 이용하여 짧은 추론 시간과 적은 에너지 소모의 장점을 갖는 스파이킹 인공신경망 설계 방법을 다룬다. 스파이킹 인공신경망은 인공신경망의 높은 에너지 소비 문제를 극복하기 위한 유망한 대안 중 하나이다. 기존 연구에서 심층 인공신경망을 정확도 손실없이 스파이킹 인공신경망으로 변환하는 방법이 발표되었다. 그러나 기존의 방법들은 rate coding을 사용하기 때문에 긴 추론 시간을 갖게 되고 이것이 많은 에너지 소모를 야기하게 되는 단점이 있다. 이 파트에서는 페이즈에 따라 다른 스파이크 가중치를 부여하는 방법으로 추론 시간을 크게 줄이는 방법을 제안한다. MNIST, SVHN, CIFAR-10, CIFAR-100 데이터셋에서의 실험 결과는 제안된 방법을 이용한 스파이킹 인공신경망이 기존 방법에 비해 큰 폭으로 추론 시간과 스파이크 발생 빈도를 줄여서 보다 에너지 효율적으로 동작함을 보여준다.

두번째 파트에서는 공정 변이가 있는 상황에서 동작하는 고에너지효율 아날로그 인공신경망 설계 방법을 다루고 있다. 인공신경망을 아날로그 회로를 사용하여

구현하면 높은 병렬성과 에너지 효율성을 얻을 수 있는 장점이 있다. 하지만, 아날로그 시스템은 노이즈에 취약한 중대한 결점을 가지고 있다. 이러한 노이즈 중 하나로 공정 변이를 들 수 있는데, 이는 아날로그 회로의 적정 동작 지점을 변화시켜 심각한 성능 저하 또는 오동작을 유발하는 원인이다. 이 파트에서는 ReRAM에 기반한 고에너지 효율 아날로그 이진 인공신경망을 구현하고, 공정 변이 문제를 해결하기 위해 활성도 일치 방법을 사용한 공정 변이 보상 기법을 제안한다. 제안된 인공신경망은 1T1R 구조의 ReRAM 배열과 차동증폭기를 이용한 뉴런을 이용하여 고밀도 집적과 고에너지 효율 동작이 가능하게 구성되었다. 또한, 아날로그 뉴런 회로의 공정 변이 취약성 문제를 해결하기 위해 이상적인 뉴런의 활성도와 동일한 활성도를 갖도록 뉴런의 바이어스를 조절하는 방법을 소개한다. 제안된 방법을 사용하여 32nm 공정에서 구현된 인공신경망은 3-sigma 지점에서 50% 문턱 전압 변이와 15%의 저항값 변이가 있는 상황에서도 MNIST에서 98.55%, CIFAR-10에서 89.63%의 정확도를 달성하였으며, 970 TOPS/W에 달하는 매우 높은 에너지 효율성을 달성하였다.

주요어: 인공 신경망, 스파이킹 신경망, 지도학습, 아날로그 뉴런, ReRAM, 활성도 일치, 공정 변이 보상

학번: 2015-30195